**TITLE**

**Automating the geographical and temporal comparison of GPX (GPS Exchange Format) files for running and walking activities**

**ABSTRACT**

When people walk, jog or run GPS enabled smartwatches and smart devices can record their activity automatically using a well-known format called GPX (GPS eXchange format). GPX is XML-based and contains a complete track or trace of their movement, usually at a high temporal resolution of two or three seconds. Many people upload their GPX tracks to social platforms for runners and hikers such as Wikilok or Strava. Despite the visualisation power of these types of tools and platforms it is still very difficult to actually compare two different GPX traces. This comparison is particularly difficult if the GPX trace represents two different activities in two different geographical areas generated by people of different fitness or physical ability. In this thesis we describe the design and development of an automated approach to compare and classify GPX traces against a database of previously seen traces. Software is written in R to automatically parse GPX files and computer a large number of characteristics related to the trace: distance statistics, time statistics, trace complexity, etc. Using machine learning approaches new and previously unseen GPX traces are then compared or matched to the most similar existing GPX traces in our database. This functionality allows people to compare a newly generated GPX trace against previously known GPX traces in order to find other GPX traces which are most similar in their spatial and temporal statistical characteristics. A simple web-based interface is provided to facilitate easy upload and insertion into the PostGIS database at the backend. The software developed in this project could potentially be used in social platforms for runners, joggers and walkers in order to quantitatively and non-subjectively compare their generated GPX traces.

**ACKNOWLEDGEMENTS**

**TABLE OF CONTENTS**

**LIST OF TABLES AND FIGURES**

**CHAPTER 1: Introduction**

**Section 1:1 Introduction to the problem**

Based on different GPX files uploaded by different users round the globe, we are trying to estimate or build up a model so that it can empower us to predict sports activities in different geo-tracks. Lots of IOT devices like smart phones, smart watches are being regularly used to capture data while performing these kinds of activities. Fast life and high expectancy of life made us serious about getting involved in fitness programs. This project is an attempt to not only predict activities for a certain geo track but also plotting an association between similar tracks. According to specific fitness goal, one can choose similar tracks, available as per his choice of city or country. As we tried to cluster tracks as per the difficulty level, the project will be helpful to choose other tracks to increase the fitness level as per the selected sport activity.

[[implement different clustering methods and find efficiency.

Predictive analysis of different sports activities. (academic paper)]]

**Section 1:2 Project Objectives**

two or three specific objectives

**Number 1: Allow the comparison of GPX files from different users and geographical areas to allow the comparison and matching of these GPX files based on their geographical and temporal characteristics.**

**Number 2: Use machine learning approaches for the purposes of comparison and matching of the GPX files**

**Number 3: Build a web-based application to allow users to upload their own personal GPX tracks. This application will automatically suggest similar GPX tracks, from the database, which have similar characteristics to the uploaded GPX track.**

**Number 4: Allow the upload of GPX tracks from any geographical location and for any foot-based activity (hiking, jogging, walking or running).**

1. Grouping GPX files as per their characteristics and prediction of activity performed suitable for certain tracks.
2. Building a web platform to enable users to choose tracks as per their fitness program. For a certain activity, the tracks will be grouped as per different level of competency levels. Users will be able to choose tracks as per their need. The selection can be made as per their neighbourhood or city or any country basis.
3. ~~The project may help various government bodies to set their priorities to sanction or complete projects in building tracks suitable for sports like running, hiking, cycling etc. If they can estimate the utilization of a prospective route and predict the kind of activities will be performed on them, will certainly help the organizations to plan for cost estimation and feature build up.~~

**Section 1:3 Overall Solution**

Not detailed specification of methodology

When we are done with

**Section 1.4 Outline of the thesis**

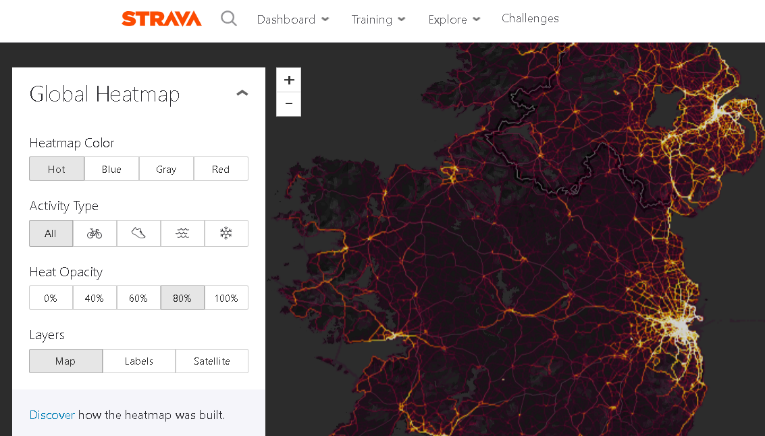
**CHAPTER 2: Background and Related Work**

**Section 2:1 Related Literature and Other Works**

**Youtube videos: gist**

This project is inspired by the academic paper, ‘Identifying the sport Activity of GPX Tracks’. It was demonstrated in the paper how 8500 GPS tracks from 10 different kinds of sports were analysed and how accurately a suitable sport activity for a certain track was anticipated. Now a days, apps like Runtastic, strava.com/heatmap are very frequently used by the common citizens around the world. These apps show how many tracks are available in a specific location. They also tell what kind of activity was performed in those tracks. Actually, these apps give user an option to upload GPX files, which got generated while being in a sport activity.

A basic Strava page looks like:

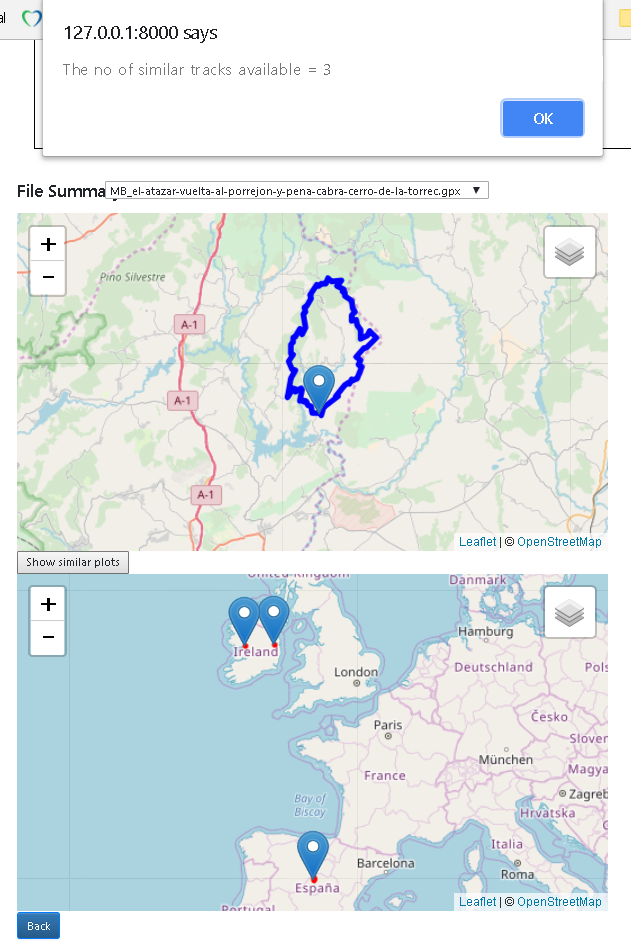


It is witnessed that map is populated as per uploaded GPX tracks.

**Section 2:2 Outline of contributions**

Now, if we could have grouped different trails as per similar characteristics and make relation between them, we could help the users to choose right tracks for their fitness programs as well. This is where our project will come into the picture.

If we utilize our project successfully and fit a large set of data, we can provide users option to make selection between similar tracks. We used Kmeans algorithm and executed the algorithm on characteristics like altitude, total-time-taken, speed, elevation-difference, total-distance-covered, avg-angle-change. For the time being all the mentioned characteristics are given with same weightage. But we can provide users choice **to set rules based upon which weightage of the characteristics will be decided** and on top of that clustering algorithm will be executed. This will make the application interactive and relevant for any user’s fitness program.



Though we are maintaining basic UI features in the project, but it is enough to depict the outcome achieved. The first map was populated as per a track selection by a user and the second map detailed all the tracks which are having similar characteristics. For this instance, only 3 similar tracks are found and prompted in an alert-box. Please note tracks grouped in the same cluster might not have similar shape as the clustering algorithm is performed on a group of characteristics. But if user impose rule for only ‘change in angle of the route’, clustered trails will have similar kinds of shape rather than anything else.

In short, in this project an attempt is made to propose a software-based solution to help citizens choosing sports tracks to enhance their fitness capability. Unsupervised machine learning concepts from data science were inherited and implemented through R and Python programming and a web-based platform is built with the help of advanced like Django framework, PostgreSQL, Jquery, Leaflet etc to provide meaningful user experience.

**CHAPTER 3: Design and Solution Overview**

To achieve the project objectives discussed in section 1.2, we came up with a process flow diagram that conjugates mainly 3 platforms: python with Django framework, PostgreSQL& R programming. Python with Django was used to set up the web platform, PostgreSQL for database management system and R for data exploration, analysis & implementation of machine learning algorithm. By using unsupervised machine learning, we tried to find similar tracks which possess homogeneous characteristics. Each of the part has its own significance and the roles of them are detailed within their own divisions. We divided the execution flow in 3 main phases. Each of they are detailed below.

A close up of a map

Description generated with very high confidence

**Data Assemble & Comprehension [1st Phase]:**

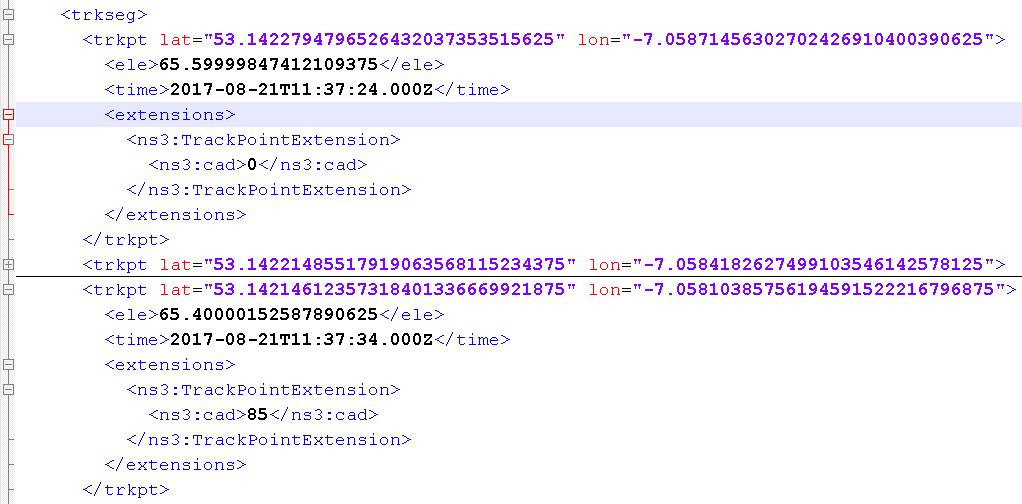
Our project work started with understanding the GPX file content details. **GPX**, or **GPS Exchange Format**, is an [XML schema](https://en.wikipedia.org/wiki/XML_schema) designed as a common [GPS](https://en.wikipedia.org/wiki/GPS) data format for software applications. ( <https://en.wikipedia.org/wiki/GPS_Exchange_Format>). The schema consists of node hierarchies. The file format is built with 3 main node types like: waypoint, track (trkpt) & route (trkseg).

*waypoint :* A waypoint simply signifies a geographical location though longitude and latitude and it has no relation with any other waypoint. It is expressed in WGS 84 (World Geodetic System) standard.

Route (trkseg): It is an ordered list of waypoints, leading to the destination.

Track (trkpt): It contains at least one track under its hierarchy.

A sample GPX file snapshot is attached below:



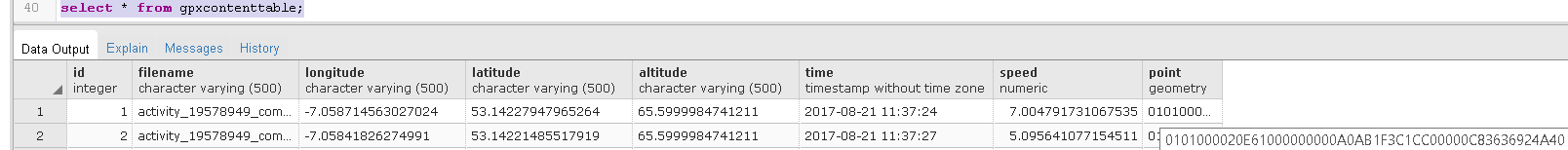
Each of the trkpt node/element holds longitude and latitude information as in attribute. Elevation(ele) and time elements provide us altitude and datetime related to a specific position.

We read the content of each test gpx file through python programming. We used ‘gpxpy’ package for pursing and fetched information like: Filename, Longitude, Latitude, Altitude, Time & Speed. The relevant code-snippet is found below.



Al the information collected is inserted in a PostgreSQL table, named ‘gpxcontenttable’. No validation rule is applied to this table. It acts as an input table and its content is utilized for future validation and analysis in R environment. Having said that, we added one more field (point) to this table. The field is of type geometry and stores combined value of longitude and latitude. We triggered below 2 lines of code for this purpose.

* *ALTER TABLE gpxcontenttable ADD COLUMN Point geometry(Point,4326)*
* *UPDATE gpxcontenttable SET point = ST\_SetSRID(ST\_MakePoint(longitude::double precision, latitude::double precision), 4326) where latitude <> '' and longitude <> '';*



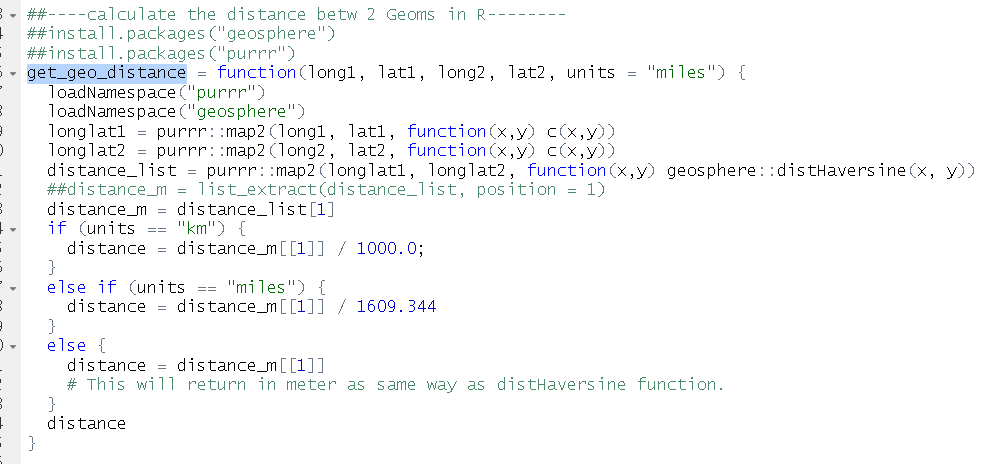
**Data transformation, enhancement & implementation of ML Algorithm [2nd Phase]:**

The 2nd phase mainly completed within R environment. The first task was to load the content of ‘gpxcontenttable’ into a list in R. We installed a package named, RPostgreSQL. By providing the DBMS and database connection details it enabled us to load the data in gpxcontent list. After performing basic data validations, like data sequence ordering as per file and time, modifying datetime format so that it can be utilized in R programming etc, we evaluated some derived fields.

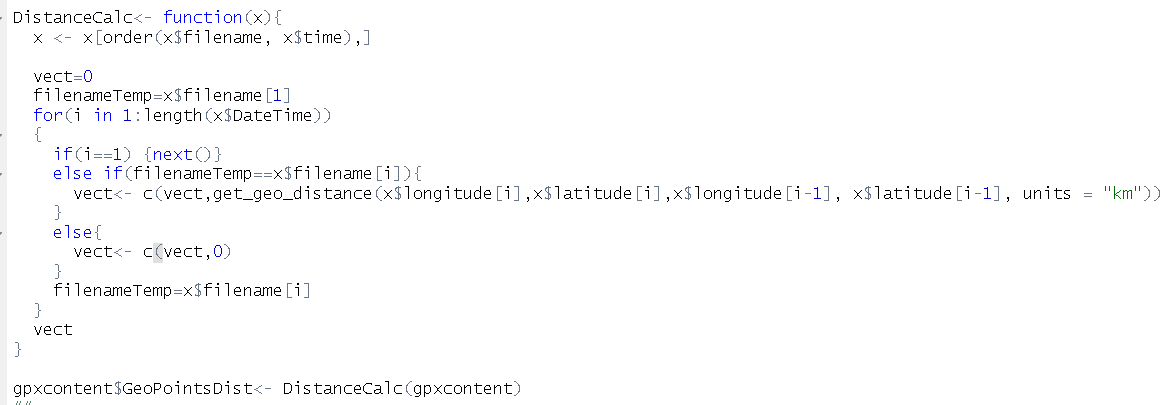
The intension of creating new derived fields are explained in grid below.



The last 2 fields derivations are complex. We were required to install 2 packages, named ‘geosphere’ & ‘purrr’ to get geo distance between 2 points. (function *get\_geo\_distance*)



Get\_geo\_distance function is called each time to determine the distance between every consecutive way-points. DistanceCalc function was written to populate GeoPointsDist field. The start and end point will always have zero in their GeoPointDist field.

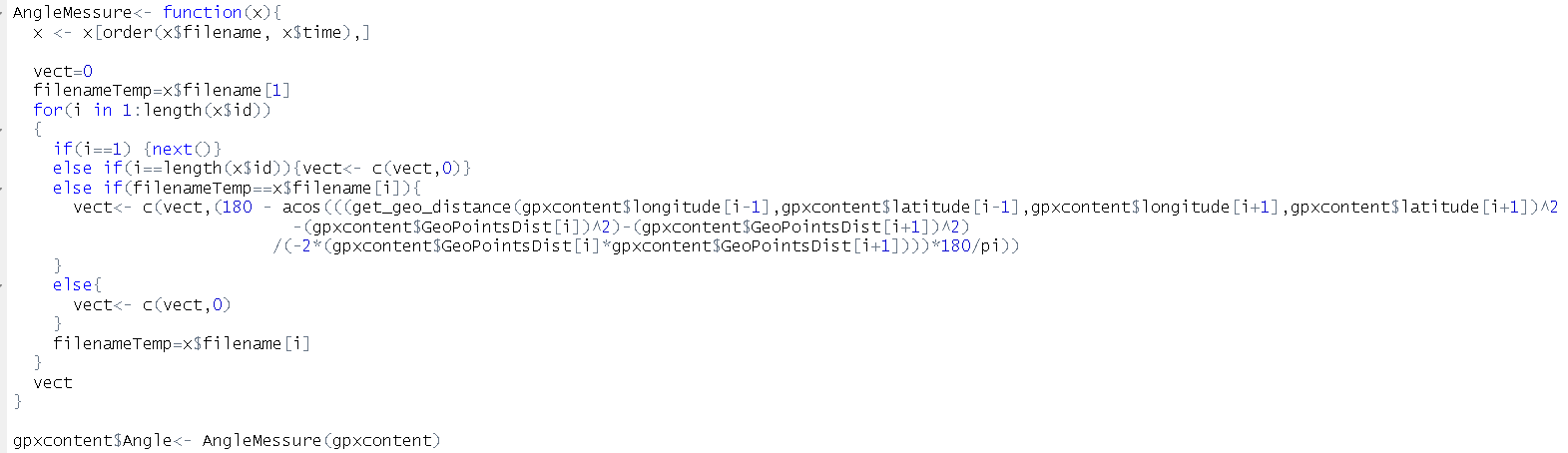


Now, we will try to understand, why we measured bend in any track. It is seen that turn present in any track has a significant role in decreasing the speed of a moving body, specially in motor related vehicles. Though in this thesis paper we are focused to test data generated from running/walking activities, angle has a significant role in making the overall clustering success.

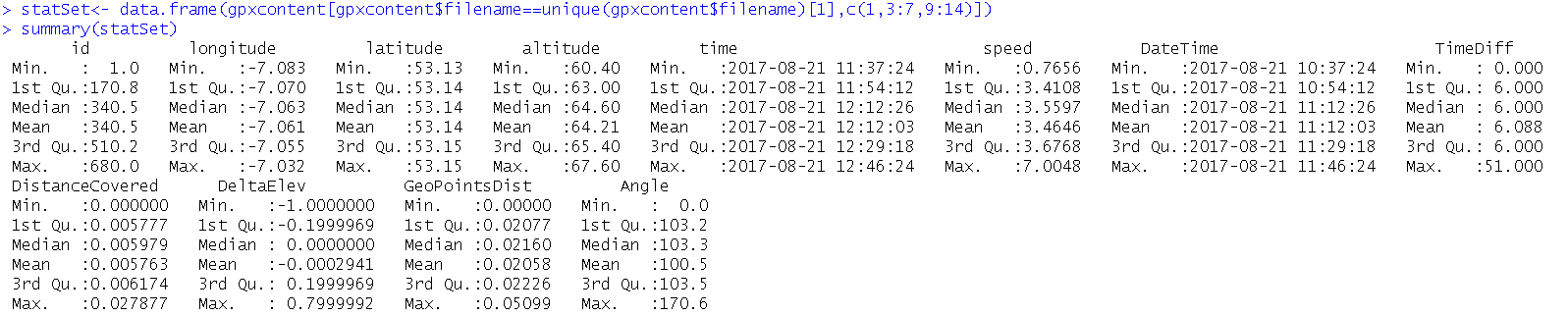
A close up of a map

Description generated with high confidence

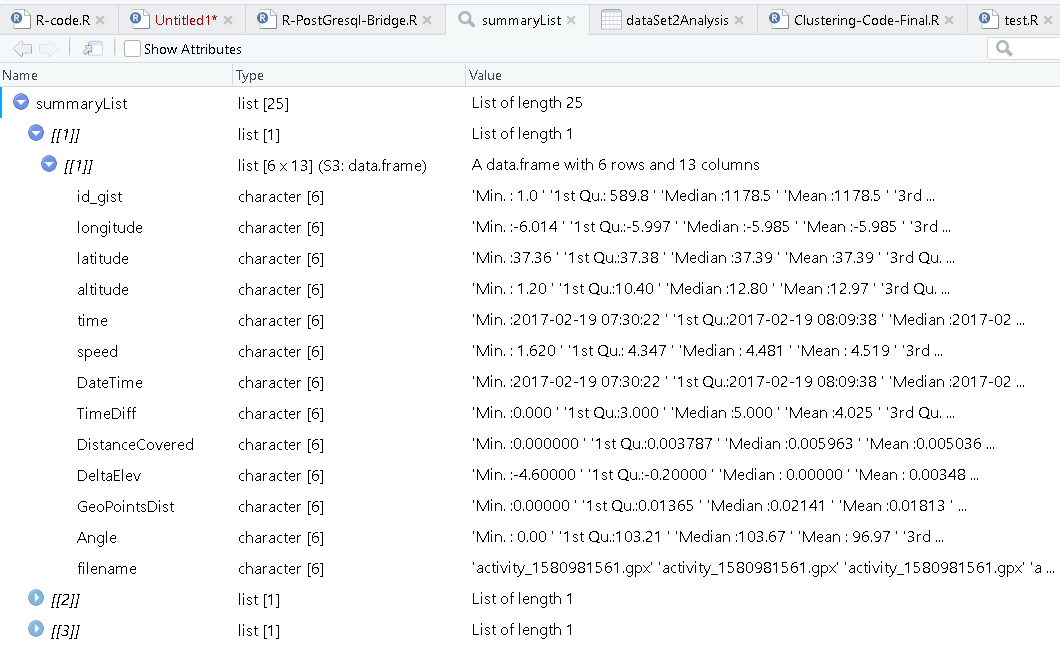
In the diagram, for an instance say we are moving from A to C via B point. If A, B & C were in a straight line the angle of B would be 180 degree. Cos 180 is equal to -1. C is 45 degree apart from the projected straight line. So, the angle present at B is 145 degree. To conclude, we can say that for any angle calculation we need to have 3 points and the middle point’s angle can be calculated (as we are tracking a line/path). The starting and end angle is measured as zero. We considered zero as well if 3 points considered are found in a straight line. The function snippet is mentioned below:



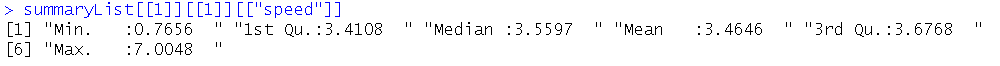
Till this point we were done with data manipulation and derivation. Now, on basis of each file we found out summary statistics and stored the result in ‘summarytable’ PostgreSQL table.



After analysis and discussion, we followed different conventions for different attributed. For speed, angle, TimeDiff we considered median value from the statistics. Whereas, for time, altitude & DeltaElev we found out the difference between their maximum and minimum value. Finally, for GeoPointsDist none of the summary statistics field helped. That is why we summed all the distance between each consecutive node or waypoint. More specifically, the dataset is actually a data frame, whose each of the entities points to different list.



That is why, for selecting certain statistical values for certain fields, we had to code accordingly. For an instance, to select the statistics related to ‘Speed’:



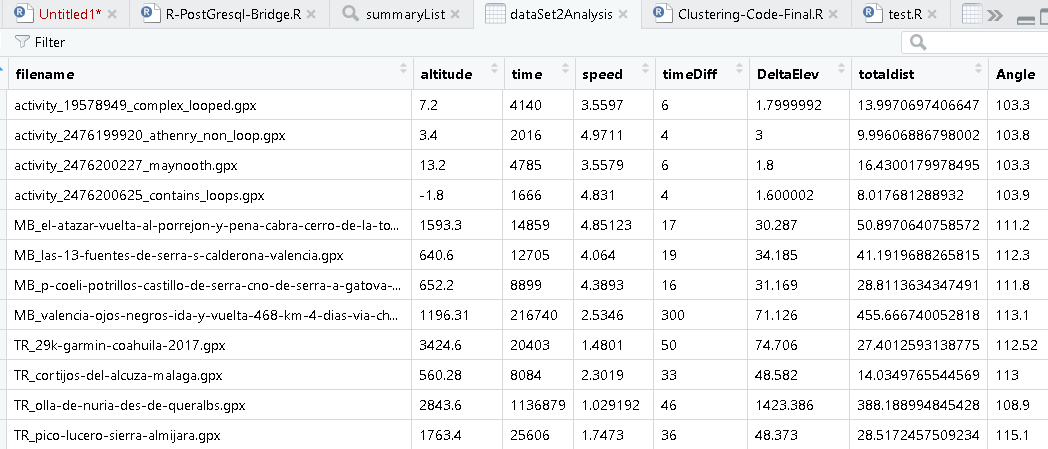
Within the 6 statistical parameters obtained, we are focused to accept 3rd value, median for further calculation.



Here we made the calculation with respect to the first file. For different files and fields, we were required to iterate though the list. For the sake of automating the whole calculation we wrote a function like:



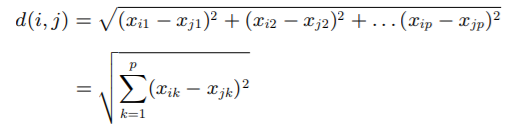
We stored the result in a dataset named dataSet2Analysis, which is of type, list in R. The content of the dataset is found as:



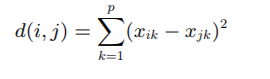
Now we have the dataset ready for various clustering algorithm to run. We mainly used a) Hierarchical clustering & b) Kmeans algorithm in ours application.

To check the similarity between two objects, we try to find the distance between them. More distant they are, less likely they are similar to each other. They are 3 conventions to find the distance. Let’s Consider case i with coordinates xi1, xi2, . . . , xip, and case j with coordinates xj1, xj2, . . . , xjp.

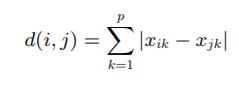
1. The Euclidean distance between case i and case j is:



1. The squared Euclidean distance between case i and case j is:



1. The Manhattan distance between case i and case j is:



For further calculation in regarding cluster, we considered **Euclidean** method for finding out distance.

a) Hierarchical clustering: If we have n objects. we wish to form k clusters. Construct a distance matrix D.

1. Start with m = n clusters.

2. Find the closest pair of clusters and merge them. Now there are m = n − 1 clusters.

3. Repeat step 2 until m = k Usually, you use k = 1.

Then, examine the sequence of partitions constructed and pick “the best” one. Step 2 above requires a distance measure between two clusters. Suppose we have two clusters U = u1, u2, . . . our and V = v1, v2, . . . vs.

Now, the distance measuring methods are of 3 types. They are:

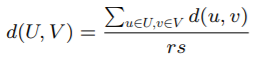
* **Single linkage distance:** When we try to find the smallest distance between a point in cluster U and a point in cluster V.

****

* **Complete linkage distance:** When we try to find the maximum distance between a point in cluster U and a point in cluster V.

****

* **Average linkage distance:** When we try to find the average distance between points in cluster U & V.

****

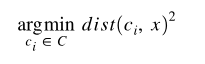
* **Distance between centroids:** Here we find distance between 2 centroids found 2 separate clusters.Let u be the centroid of cluster U and v be the centroid of cluster V.

****

**b) K-means Clustering:** The *Κ*-means clustering algorithm uses iterative refinement to produce a final result. The algorithm inputs are the number of clusters *Κ* and the data set. The data set is a collection of features for each data point. The algorithms starts with initial estimates for the *Κ*centroids, which can either be randomly generated or randomly selected from the data set. The algorithm then iterates between two steps:

1. Data assigment step:

Each centroid defines one of the clusters. In this step, each data point is assigned to its nearest centroid, based on the squared Euclidean distance. More formally, if *ci* is the collection of centroids in set *C*, then each data point *x* is assigned to a cluster based on



where*dist*( *·*) is the standard (*L*2) Euclidean distance. Let the set of data point assignments for each *ith* cluster centroid be *Si*.

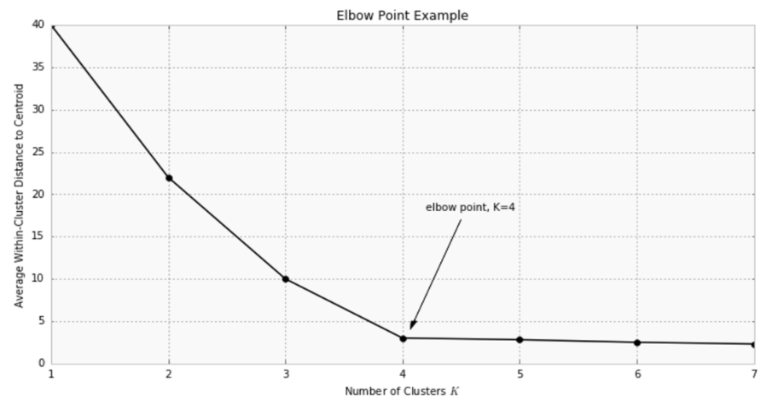
2. Centroid update step:

In this step, the centroids are recomputed. This is done by taking the mean of all data points assigned to that centroid's cluster.



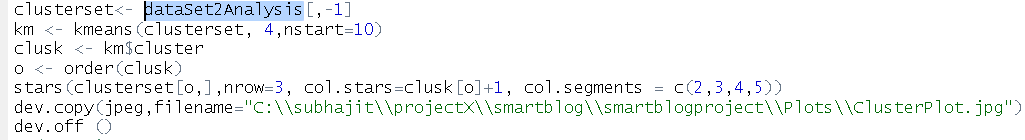
The algorithm iterates between steps one and two until a stopping criterion is met (i.e., no data points change clusters, the sum of the distances is minimized, or some maximum number of iterations is reached).

One of the metrics that is commonly used to compare results across different values of *K* is the mean distance between data points and their cluster centroid. Since increasing the number of clusters will always reduce the distance to data points, increasing *K* will *always* decrease this metric, to the extreme of reaching zero when *K* is the same as the number of data points. Thus, this metric cannot be used as the sole target. Instead, mean distance to the centroid as a function of *K* is plotted and the "elbow point," where the rate of decrease sharply shifts, can be used to roughly determine *K*.

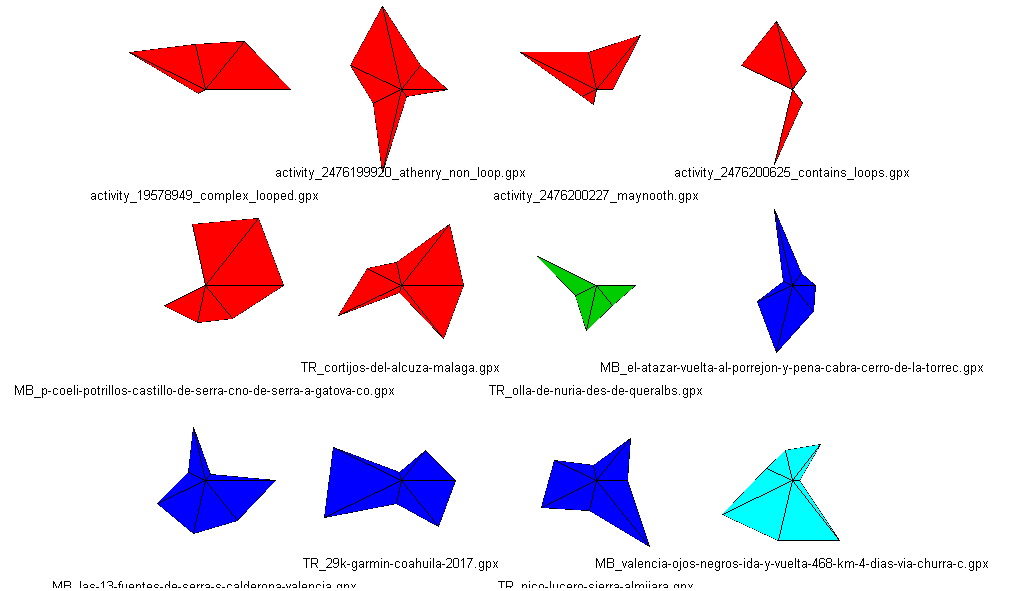


After 4th point the curve became almost parallel to the x-axis. That means for further grouping of the dataset will make the TWSS tend to zero, which is not required. So, making 4 clusters from the figure mentioned above would be most relevant.

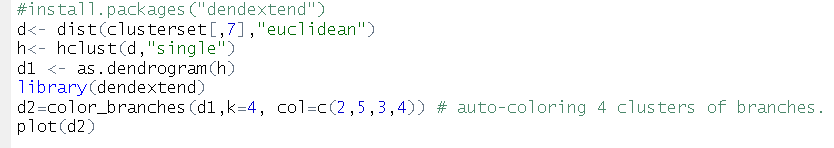
We applied the K-means algorithm on dataSet2Analysis dataset and the sample code snippet is like:



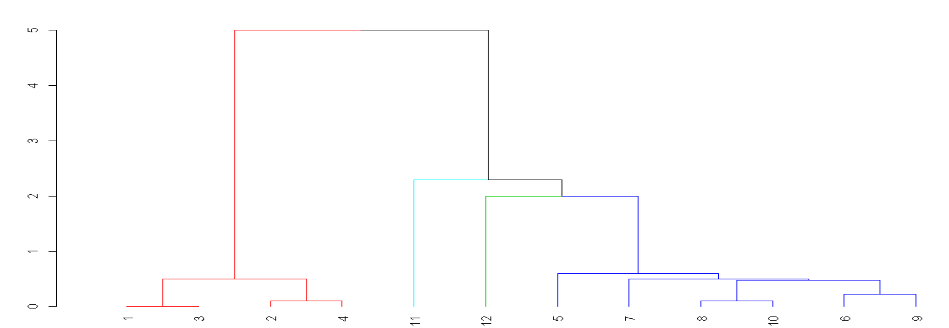
The clustered output was generated in star like images and pulled in webpages at later point of time.



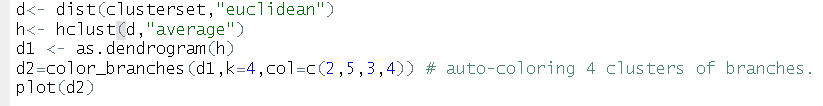
We also generated output using hierarchical clustering. The sample code, using single linkage distance:



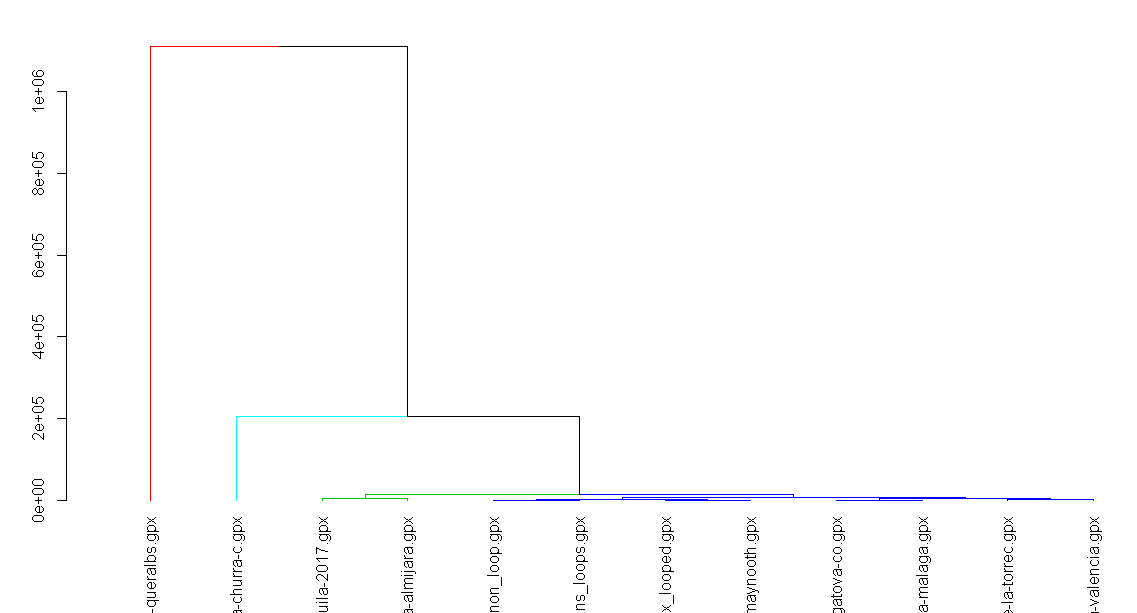
The generated output was:



While performing clustering using average linkage distance, the code was like:



And the generated output was like:

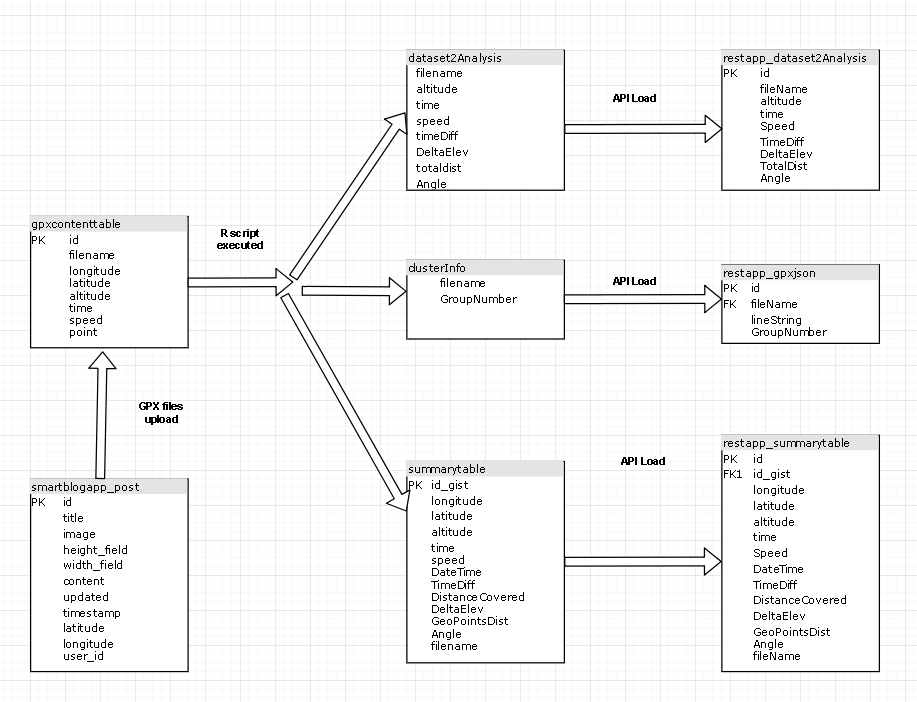


**Analysed Data Store & Output Visualization in Web platform [3rd Phase]:**

The data science related main calculation and manipulation were done in the phase 2 of the application process life cycle. As per the generated output, the data is uploaded to the respective PostgreSQL tables and generated output images are stored in respective directories. These tables and directories were referenced from the python web application to show the output in the user interface.

3 tables were bounded with model to form the REST-API. The respective tables are restapp\_dataset2Analysis, restapp\_gpxjson & restapp\_summarytable. Let’s find out how data is been stored in various phases and helped to build REST-API through a data model diagram.

**Data model diagram:**

****

The data model diagram of our project can be categorized into 4 main divisions. The first step of the application was to create a post as per activity, that is the point when smartblogapp\_post table is utilized. When the user uploads all the available gpx files generated from a certain activity for analysis from the web app, all the content is upload to gpxcontenttable. There after user is required to trigger the R script manually for data extrapolation and clustering algorithm (Kmeans) execution. In the 2nd step, the R script automatically populates dataset2Analysis, clusterInfo and summarytable with result sets. The mentioned 3 tables content at last is migrated to restapp\_dataset2Analysis, *restapp\_gpxjson* & *restapp\_summarytable* respectively, so that it can be accessed through REST-API.

The line-string field contains series of geolocation points, which accumulatively helps to create track for visualization. A sample query used in the application to form the line-string is mentioned below:

*insert into restapp\_gpxJson (id, "fileName", "lineString") select (ROW\_NUMBER() over (order by r.filename)), r.fileName, st\_asgeojson(r.Route) from (SELECT St\_MakeLine(point) as Route, tab.filename FROM (SELECT point, CAST(time As date) as Data\_obs, filename FROM gpxcontenttable as gc ORDER BY gc.time) tab group by tab.filename) as r;*

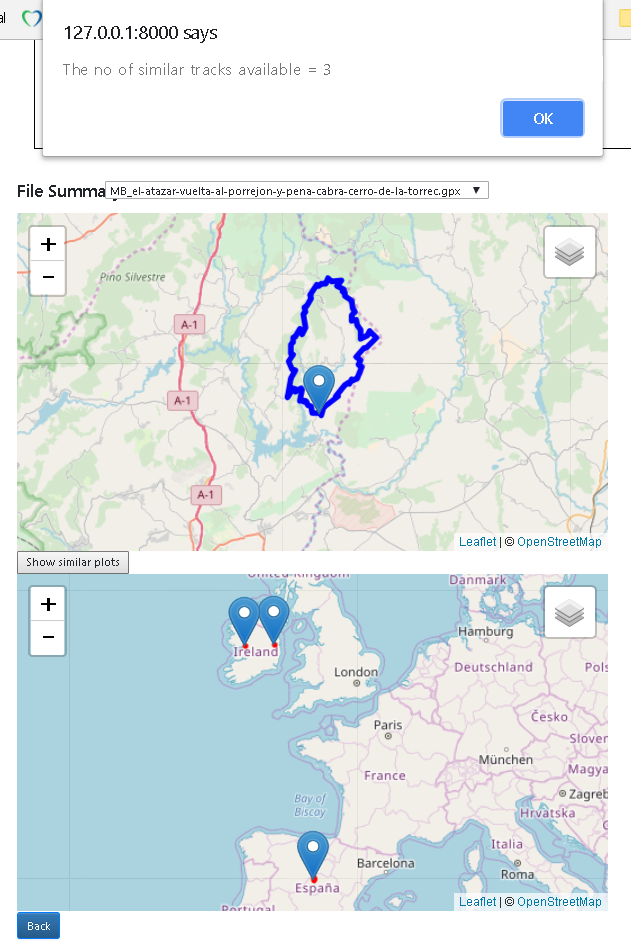
We collected the latitude & longitude information from *gpxcontenttable* tableto form a linestringand thenpopulated *restapp\_gpxJson table.* We used leaflet for visualization purpose. After consuming REST-API response, within ajax function javascript object is turned into json object and injected to Leaflet template.

**CHAPTER 4: Analysis and Evaluation**

**Describe the type of test data you have used – [35 GPX track taken from running/jogging activities in Ireland, USA and Spain. We purposely choose GPX files which were generated in different geographical areas and five people generated these files. We believe that the input set of GPX files represents a sufficiently diverse dataset for our testing and evaluation.**

As discussed in chapter 2, this project is highly influenced by the academic paper, ‘Identifying the sport Activity of GPX Tracks’. The paper was referred as a starting point and throughout the project, we tried to build a software solution where sports activity is predicted accurately, and similar tracks are clustered in same group through data science algorithm.

In chapter 3, we discussed how configured solution architecture and built data model. The built web-app is built in Django with python programming. Having said that, we required PostgreSQL, Javascript & R coding as well. As out prime objective was to find out similar tracks with in any given set of data, we are able to visualize the similar paths in the web app using Leaflet framework. The most relevant snapshot, which we can refer as the output of our project, is attached below.



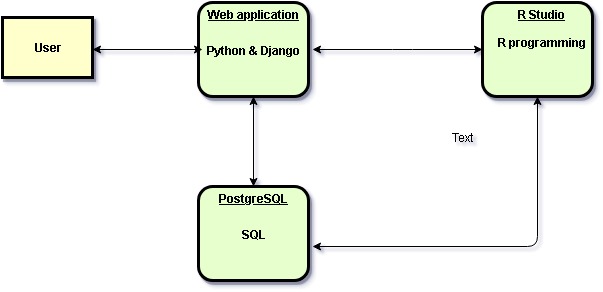
As discussed in the solution overview module, we utilized 3 platforms mainly: Python, PostgreSQL & R. The main chunk of data filtering and exploration code is written in R. R is syntactically easy to use reduced the development time significantly. But it is tested that python written code provides same kind of functionality and it boosts performance specially in software life cycle. At the later part of the development cycle, attempts were made to migrate some functionality from R to python platform. That is why currently, the ‘Elbow-curve’ generation, which has a significant role in deciding the number of clusters possible in given dataset, is getting triggered from python environment. It would be better if we could have written all the data analysis related code in python itself. Though for implementation of supervised machine learning concepts R programming is based suited, as for same kind functionality we are required to build comparatively huge chunk of codebase. That might hamper the maintainability of the whole code base. We need to find the optimum point where we can find the perfect balance between application performance and maintenance.

For the time being the triggering of R script is manual and we need to find a way around to resolve this problem.

**Application architecture:**

Our prime focus was to create a web application, so that users can visualize their GPX file characteristics, specially the route in a form of a map and choose similar routes as per their fitness program. A basic option is provided to upload all training data from the local directory to the application domain. The main 3 platforms, getting used in the application are 1) python programming with Django web framework, 2) PostgreSQL, and 3) R programming. Having said that, some other technologies or conventions like REST-API, Ajax, Javascript, Jquery library, Bootstrap library etc played a significant role in making our project a success. They will also be discussed as per the need.

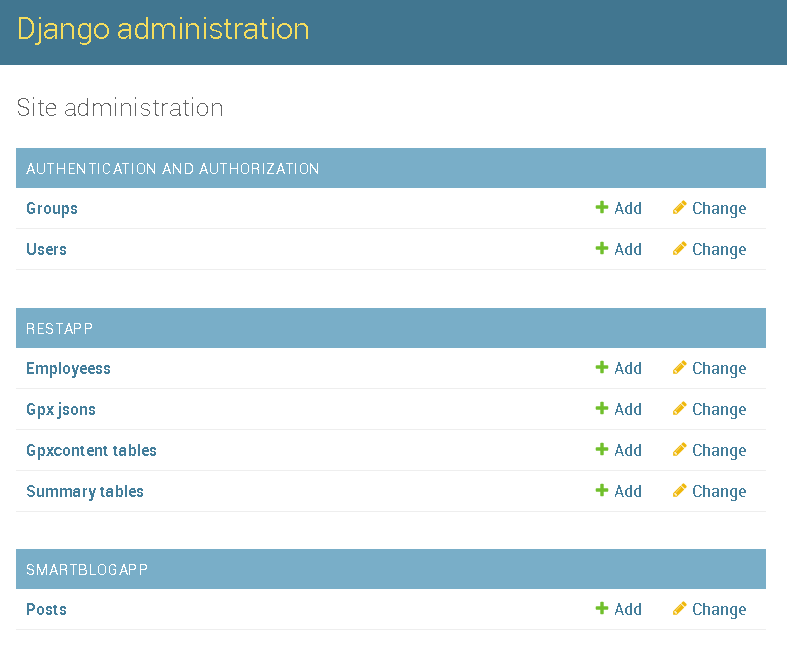
Figure:



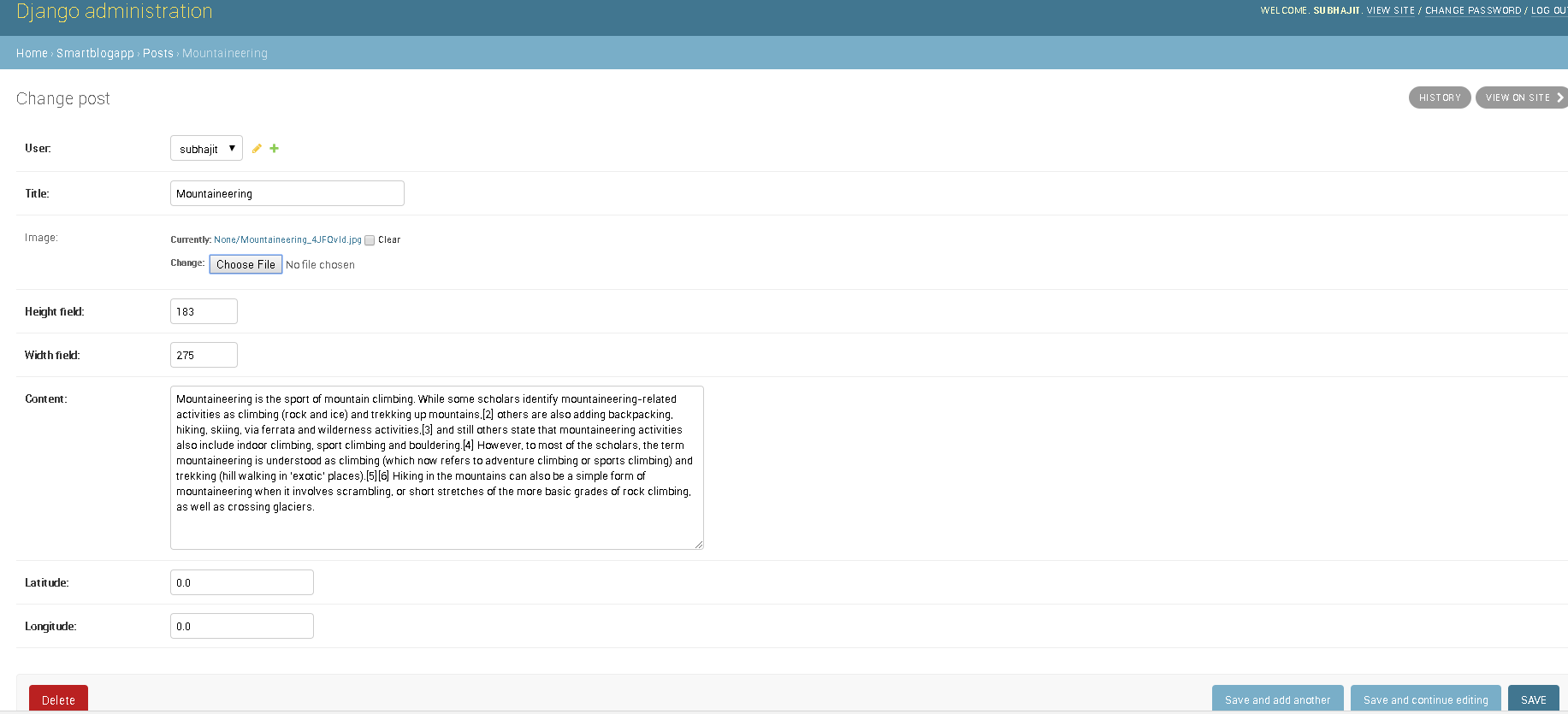
A web application is created with python programming and Django web framework. A user interacts with the application through a user interface. The web application follows client server architecture and as per need interacts with database (PostgreSQL) and executes R script. The database is maintained mainly for data store and retrieval purpose though some calculation part occurred in PostgreSQL as well (it will be detailed in the PostgreSQL section). Whereas the created R script, is triggered (for the time being the triggering is manual) from the Django framework when all the GPX files content is loaded in the PostgreSQL tables and there is a need to prepare dataset for machine learning algorithm execution. After R script execution, user will be able to load PostgreSQL respective tables, from which results will be shown up in the UI (user interface).

**Web Application:**

We created a blog application, where users will be able to upload their GPX files as per their performed activity and will be able to find out association between different tracts as per different activities. For the time being, we considered that the files get generated from one person. Later point of time data model can be customized to store records for different individuals. For the sake of application management and maintaining application model hierarchy, we created 3 apps within the same project. They are a) admin, b) smartblog-app and c) rest-app. The admin section is good enough to create a user and to provide the basic authentication and authorization for different users.



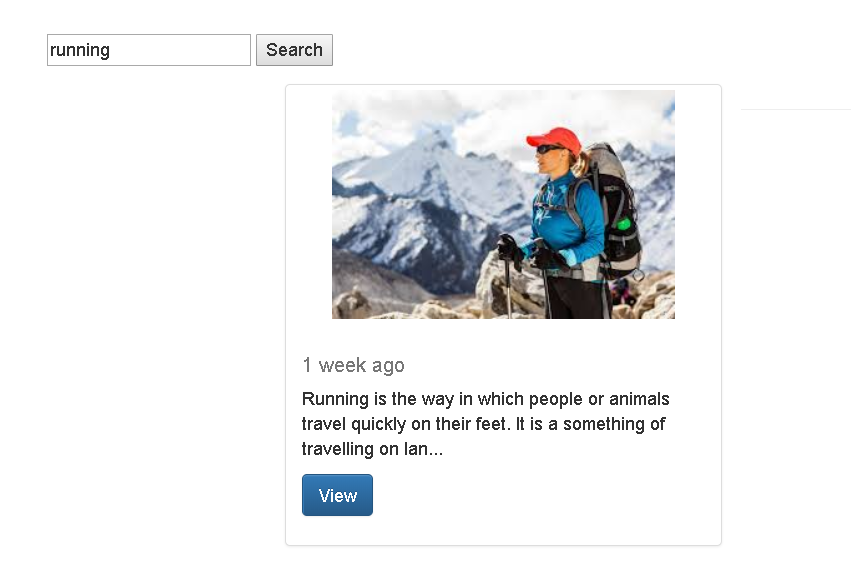
From the admin section, it will always be possible to control the entities listed within the mention applications. For an example running related information can be customized or created from the admin section itself.



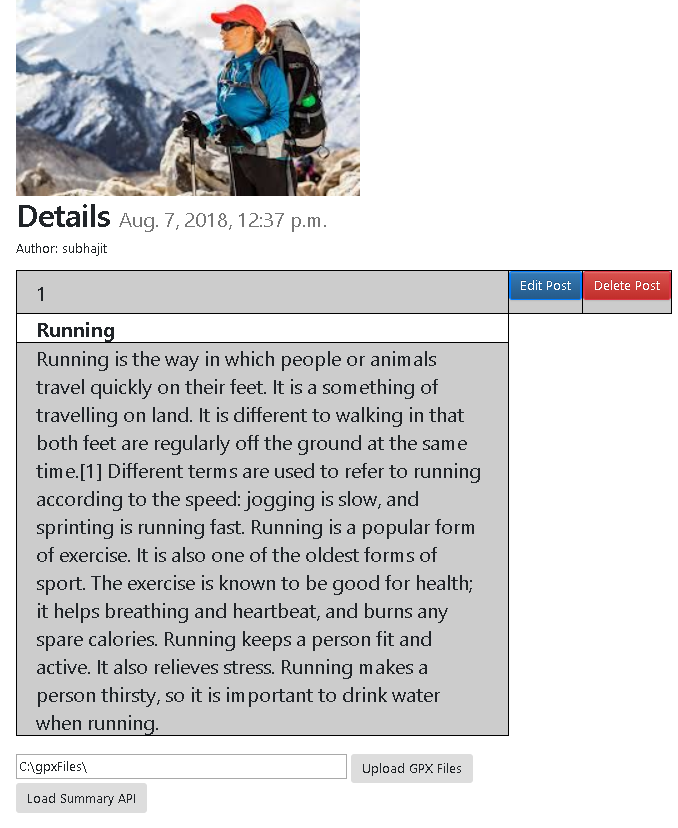
Similarly, the entities created in the rest-app can be altered from the admin section by the same process. But for each of the cases creation and alteration are manual.

This first section will also help the administrator control the main application and the endpoints that are created for REST-APIs.

The second section is the main application which holds the activity name, its description, and the creation time details. User can upload a picture relevant to the performed activity as well. The landing page looks similar to this:

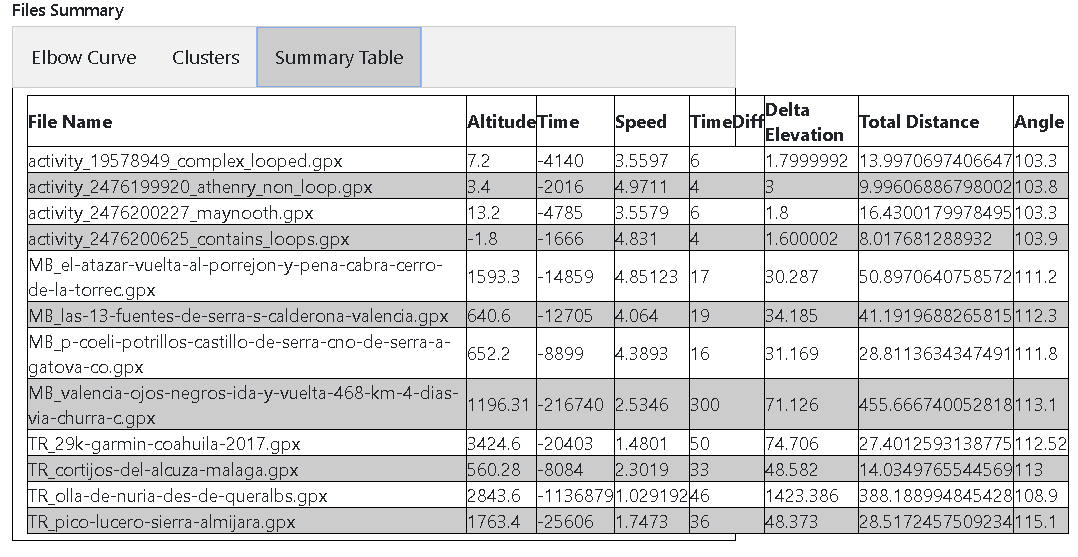


As per the entries made to the system, we can search an activity and navigate to its details page by pressing view button. The detail page will look like this:

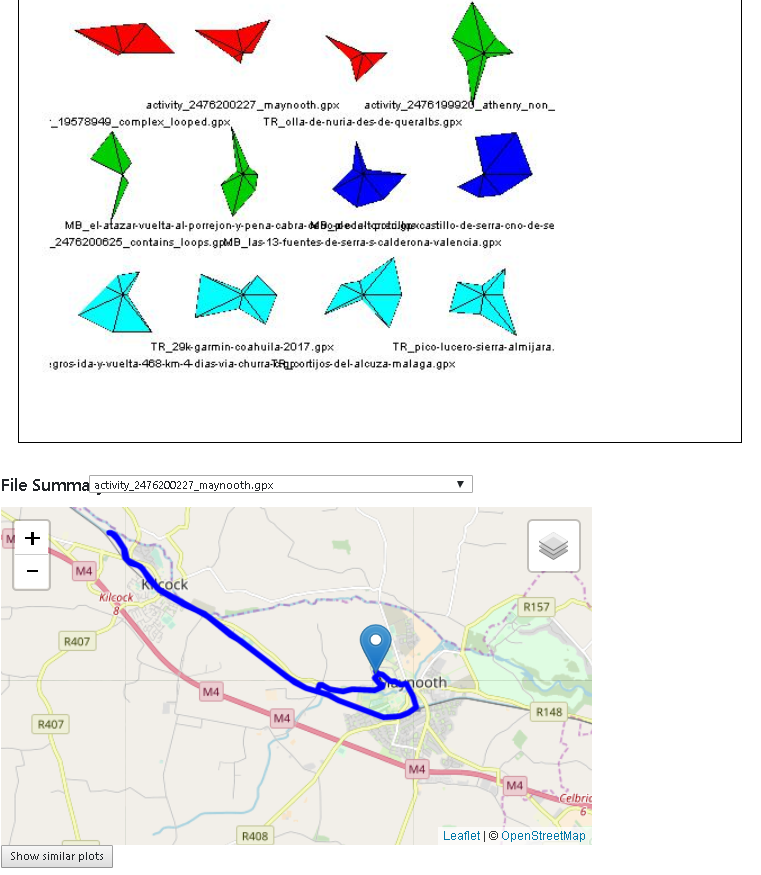


Along with edit and delete options, user will be asked to upload gpx files through ‘Upload GPX Files’ button. Please note the files should be placed under ‘c:\gpxfiles’ directory. Actually this button uploads all the content of the gpx files to gpxcontentable in PostgreSQL for further process. Now when the upload will be completed, the system will ask the user to manually run the designed R script (this step could have been automated, but faced challenges, which is described in section). After R script execution user is required to enable the all configured REST-APIs within the application to populate data as per R script output. The ‘Load summary API’ button will exactly do the same. Based upon the output generated from the R script, results are shown in different tabs.

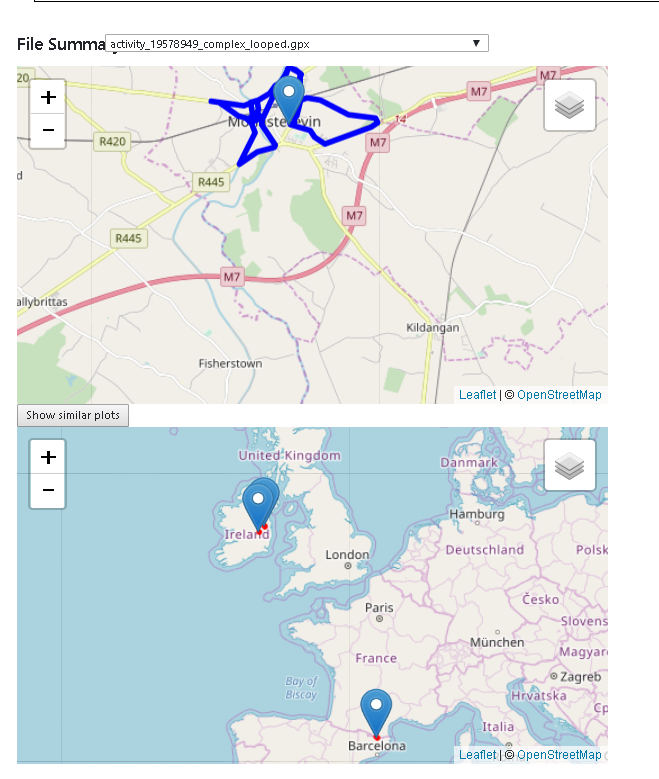




User will be able to visualize characteristics of different files find out similar tracks by colour matching star figures. All the file options will be made available to the user in a dropdown control. On selection of a track user will be able to find out similar tracks by triggering ‘show similar plots’ button.



For an example, if user selects the file ‘activity\_2476200227\_maynooth.gpx’, two other files (marked in red) should be displayed as in total there are 3 red marked files present in the star plot.



As per the test data, only 3 similar files are shown in the similar plot. In this way the application helps the user to find out same kind of tracks found across different geographic location to achieve their fitness goal.

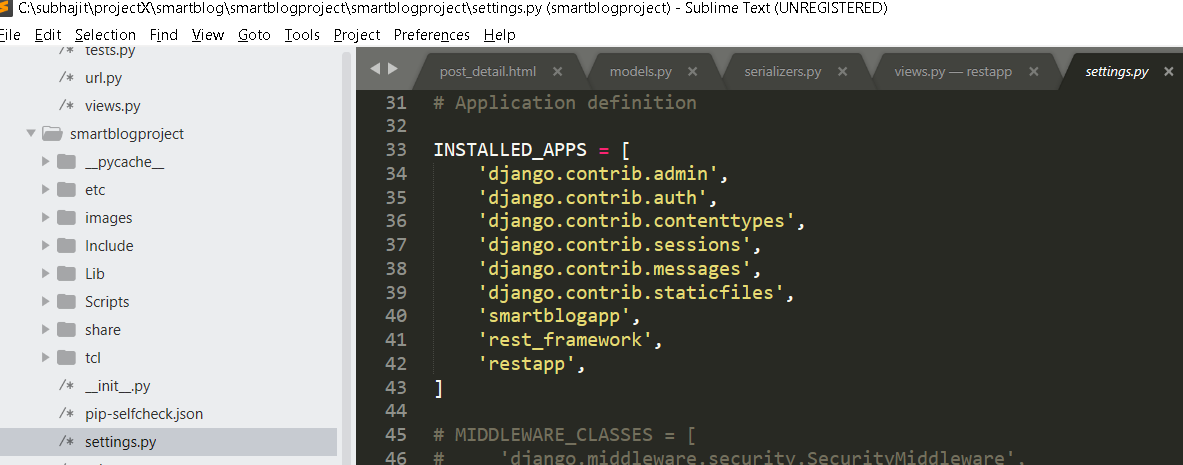
User will be able to store the relevant data and execute machine learning algorithm as per different activities. The above example was regarding ‘running’ activity and the relevant files were process.

The last part of the web application is REST-APP. This app acts as a service and hosts couple of endpoints which follow REST protocol and whichever application consume this service will fetch response in JSON format. The REST-APP fetches data from couple of tables in PostgreSQL and expose the data as in json response. In the main application ‘smartblogapp’, javascript code is written to make some AJAX calls from the client side and to consume the json response from the REST-API created.

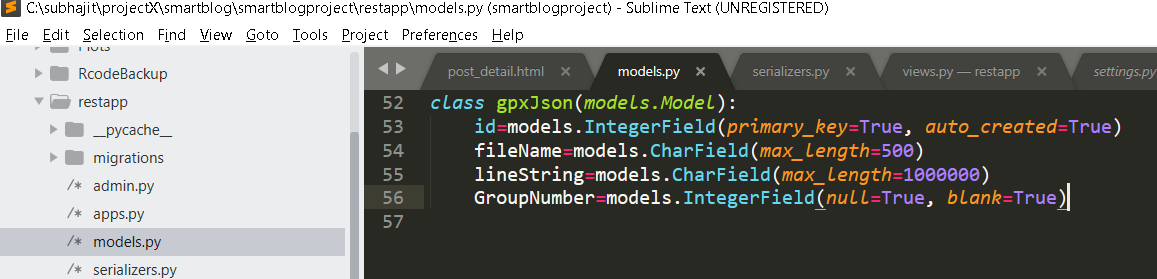
Creating a REST-API:

There are couple of steps required to create a REST-API in Django framework. They are:

1. Django-restframework should be installed using command prompt and ‘rest\_framework’ should be referred as installed app in the settings file.

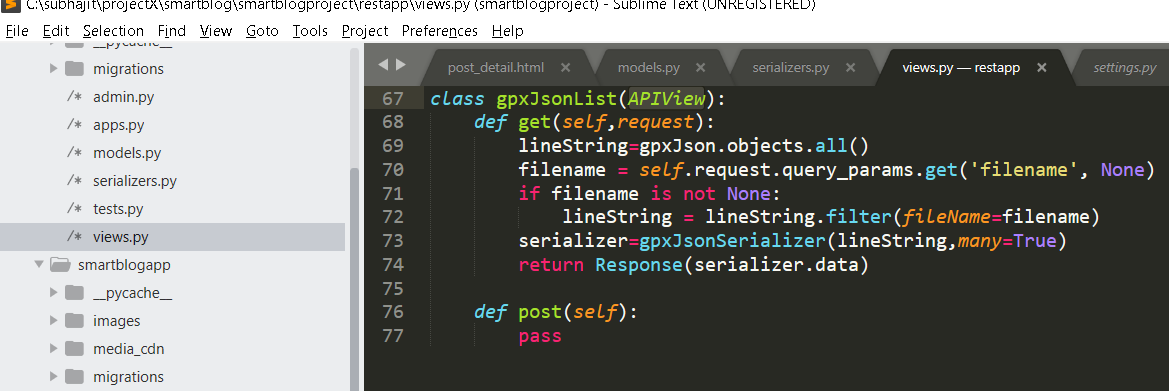


1. Create separate model class for each endpoint. Please note that the id field is reserved to make relationship between database entity and model object. Except id field, other fields are declared within the model class to map against the columns of the database table(entity). For an example while creating gpxjson class in model fields were declared like this.



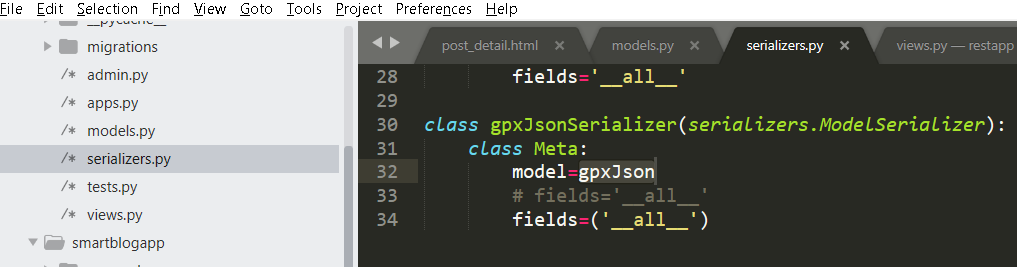
The motive was to pull line-string (that will help to draw the path) and group number (which cluster the file belongs to) with respect to filenames.

1. Create a view class for each model class created. The response of the view class should be a APIView (type) which derived from ‘rest\_framework.views’.



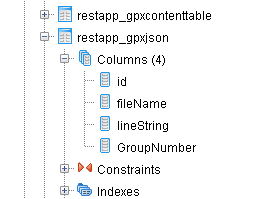
Here ‘gpxJson’ is a model object. By using query-string all the objects are mapped to lineString variable. The serialized response from the view actually exposed through an endpoint.

1. Creating serializers fills up the bridge between view and model entities. Serializers help to convert model objects into stream and vice versa.



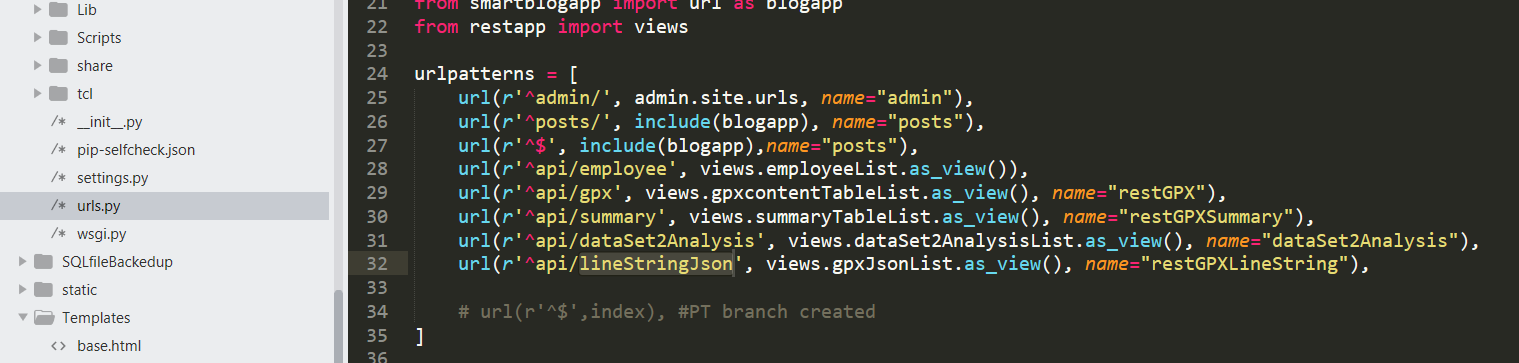
For serialization of gpxJson model entity, gpxJsonSerializer class is created for both way parsing process. Note: here all the fields are parsed with no filter restriction. By specifying only the allowed fields could have imposed filter restriction.

1. Model migration: when we are done with model changes, we can migrate changes to the database and can alter or create schema of the database entities. We are following code first approach and migration with the help of manage.py file helped us to create table in PostgreSQL like:



Note: gpxjson is the model class name and restapp is the app name. SO, by convention restapp\_gpxjson table is formed.

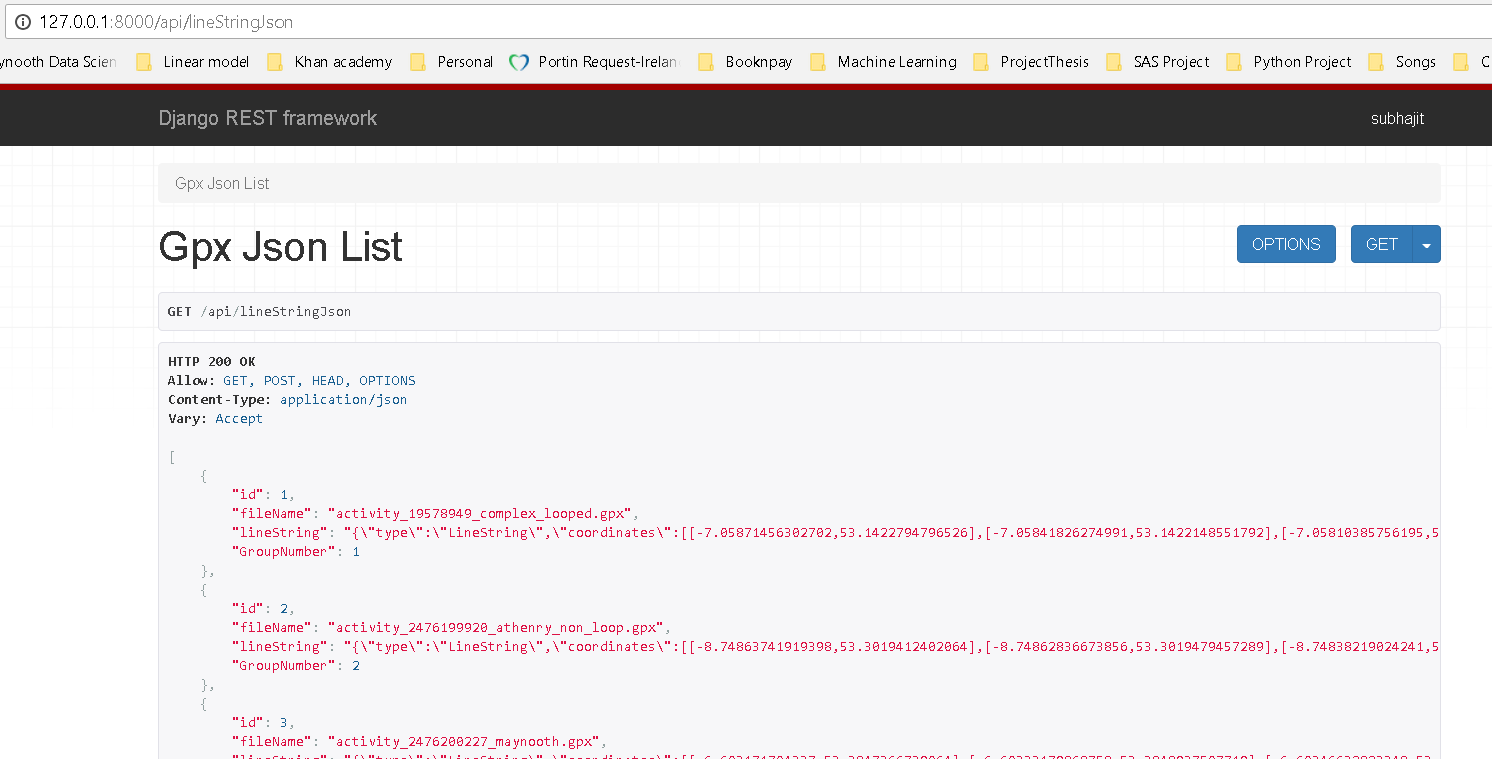
1. The last step is to expose the endpoint through a URI. In our case we defined that in url.py file like:



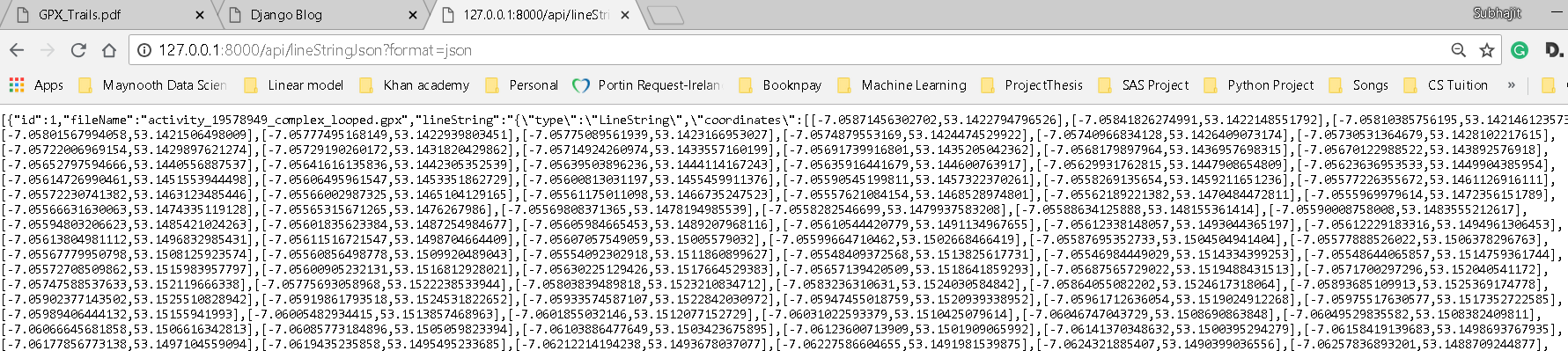
Here the last command in urlpatterns, exposes service to get linestring data along with filename and group(or cluster) number at URI <http://127.0.0.1:8000/api/lineStringJson>. Note: till this point of time it is locally configured.

Sample response of a rest-api:

We are able to achieve response in 2 modes. One is in api mode like:



And another mode is in json like:



The screen shots are attached with respect to the linestringjson endpoint. Similar responses are achieved through dataset2analysis & summary endpoints.

The whole purpose of creating REST-APIs were:

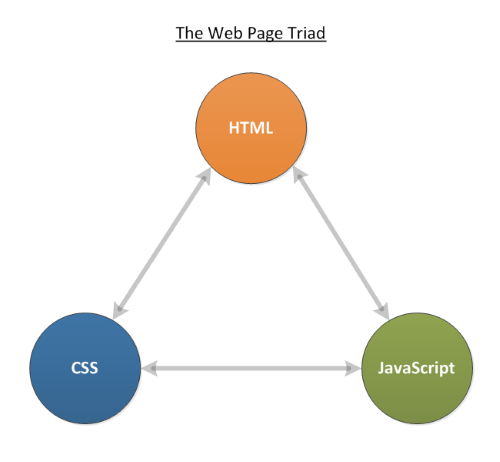
1. The data (in json format) can be easily accessed within the project as well as it can be consumed from external sources.
2. The approach made the application light weight and distributed in nature.
3. This made the modules decoupled and made the debugging process easy.

Now that we successfully created APIs, they need to be consumed and data should be reflected in the user interface. For this purpose we used Javascript, Jquery, AJAX & Leaflet. Prior to understand how they are getting utilized, lets have an brief description about each of them.

**Javascript**: It is a scripting language, that is very much efficient for client side programming, though now a days it is used in server side coding (e.g: NodeJS) as well.  It is supported by most web browsers including Chrome, Firefox, Safari, internet Explorer, Edge, Opera, etc. Most mobile browsers for smart phones support JavaScript too.

It is primarily used to enhance web pages to provide for a more user friendly experience. These include dynamically updating web pages, user interface enhancements such as menus and dialog boxes, animations, 2D and 3D graphics, interactive maps, video players, and more. This mode of JavaScript usage in the web browser is also referred to as **client-side javascript**.

When you consider the components that make up a web page, JavaScript forms the third component of the triad, HTML and CSS being the other two. [**HTML describes the page**](https://www.makeuseof.com/tag/simple-html-code-learn-minutes/), including the text, graphics, etc. CSS is used to control and customize the look of the web page, including the colors, fonts, etc. JavaScript is used to add a dynamic component to the web page and make most elements on the page programmable.



The 3 components HTML, CSS & JS compositely provide user experience. The advantage of using JS is that it can manipulate DOM (Document Object Model) at runtime. Through event handlers we can alter DOM objects without the need of server side interaction.

**Jquery:**

[**jQuery**](http://jquery.com/) is an extension of Javascript, a library of **[Javascript](https://www.makeuseof.com/tags/javascript/)** functions and utilities that add visual flair, and make advanced features simple to implement in just a few lines of code. It is supported across browsers, and open source. Plus you can also extend jQuery functionality with easy to use plugins.

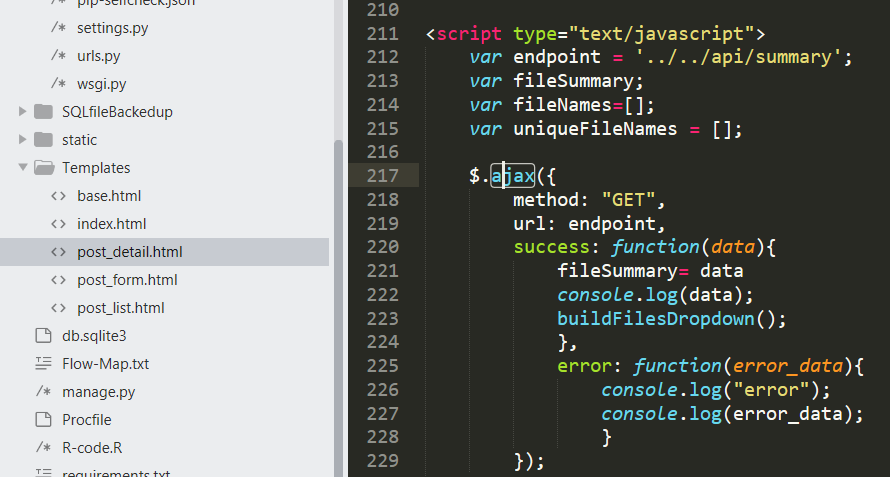
jQuery is built on top of Javascript, but it adds methods and functionality not found in pure Javascript. It was created in order to simplify Javascript usage and remove the hassles concerned with different Javascript implementation in different browsers. With jQuery you can just write once, and jQuery will interpret your code correctly for any browser.

The main function of jQuery is for DOM manipulation (DOM is the Document Object Model), and it means the underlying structure of any webpage you visit. Every single thing you see on the page – and many you don’t see – are represented in the DOM.

jQuery is not the only Javascript library out there, but it is the most popular – about 55% of the top 10,000 websites use it. [*Prototype*](http://prototypejs.org/)*, [MooTools](http://mootools.net/" \t "_blank) and [Scriptaculous](http://script.aculo.us/" \t "_blank)* are popular alternatives, but not nearly as well supported.

Jquery is used for the below mentioned advantages:

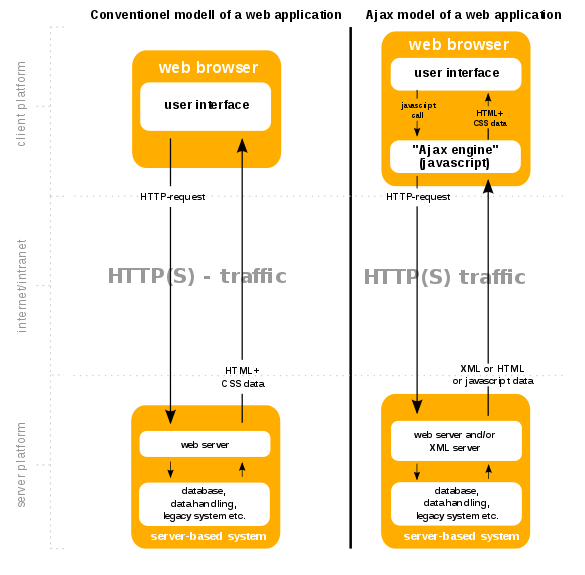
1. Defines Easy events: Nearly all software works on an event model – click on something, and a click event is triggered. Drag your finger across a tablet, and a drag event is triggered. Applications “listen” for these events and do something – jQuery lets you do this in a browser.
2. **Perform AJAX Simply:** Using JQuery, the implementation of AJAX becomes very simple. Jquery library is addressed by $ and its functionality is accessed. ‘ajax’ is one of the function that is defined within Jquery/$. We just need to pass some property values to get response from an API. A sample ajax method that was used within the project is mentioned below:

****

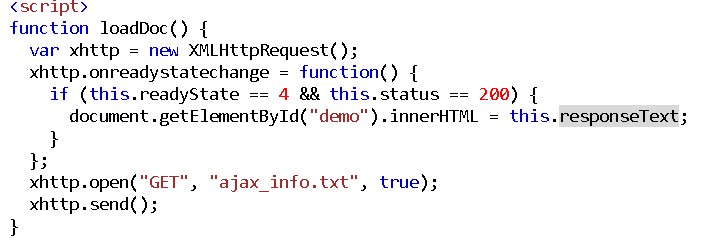
In the screenshot, method, url are the properties of ajax function. In this example we were looking for the GET web method and url was specified as per resource availability. Success and error are 2 events against which specific functions were written. For success (status code 200) we consumed the data and used it within the page DOM objects.

**AJAX** (Asynchronous JavaScript and XML):

It is a technique rather than a programming language to access web servers from the client side (web page). The whole process starts with instantiating XMLHttpRequest javascript object and creating functions around it. There are certain benefits of ajax introduction in a webpage life cycle and that is elaborated using the following diagram. (https://en.wikipedia.org/wiki/Ajax\_(programming))

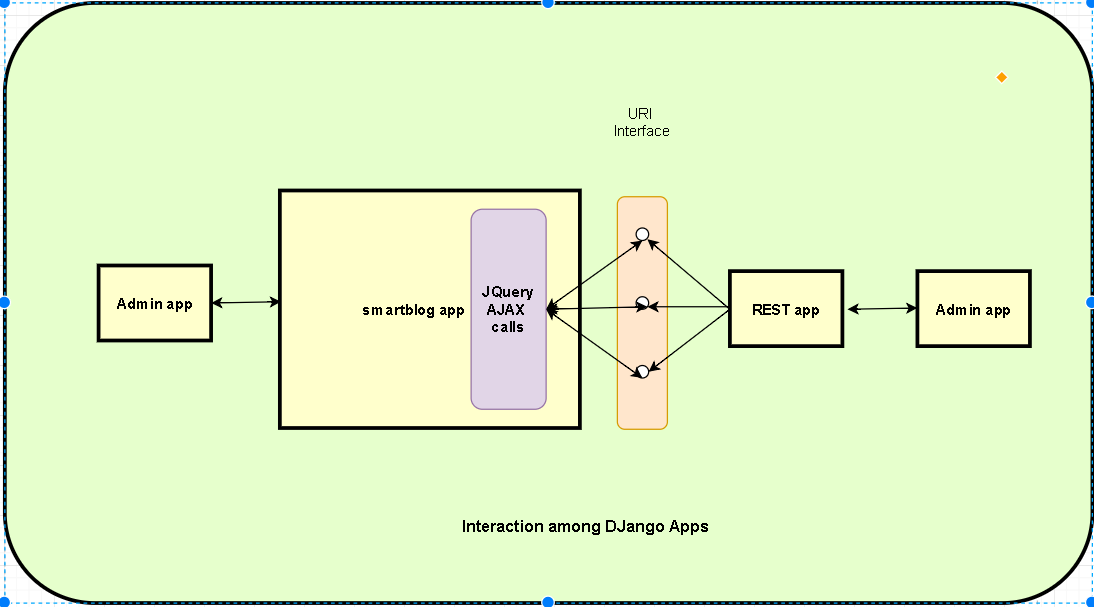


A sample AJAX code is found below: (https://www.w3schools.com/xml/ajax\_xmlhttprequest\_response.asp)



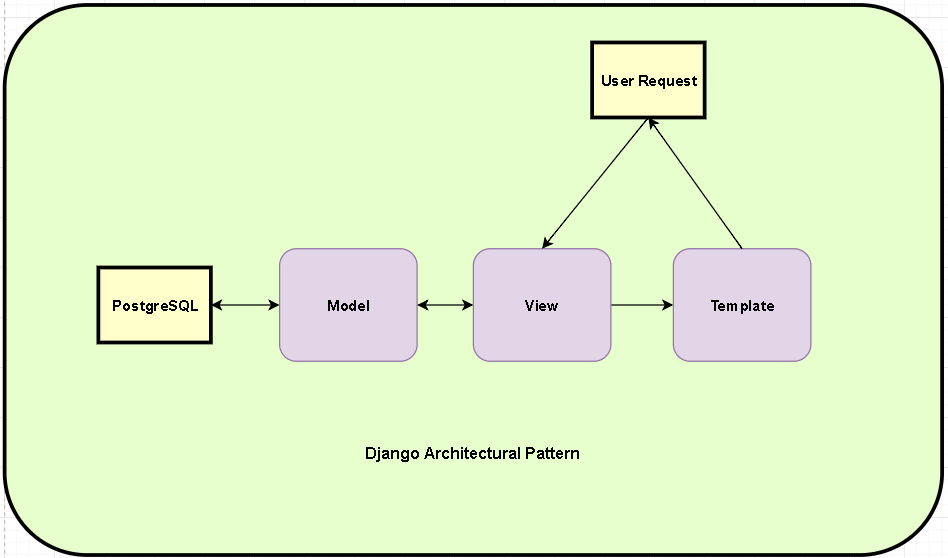
The status code 200 signifies success in securing a response from the web server. It is evident that we could have achieved same functionality either using XMLHttpRequest or by using $.ajax function, found in Jquery. We used $.ajax function as it is simpler.

**Interaction among Django apps:**



The Django project compresses 3 apps within itself. Admin, Smartblog & Rest apps makes the interaction among themselves to provide the overall functionalities. Both Smartblog and Rest apps are customizable from the Admin app. The Rest app creates an URI interface so that certain database entries are serialized and exposed to the other application domain. The main 3 URI, created are summary, data2analysis & LineStringJson. 3 of them expose the content of three tables restapp\_summarytable, restapp\_data2analysis & restapp\_gpxJson respectively. The importance of the tables will be specified in data model section. The smartblog app is built with couplt of templates (nothing by HTML pages). Within each templates javascript/Jquery code is written to consume rest-response as per requirement and reflect that in the UI.

**Django Architectural Pattern:**



Django follows the MVC pattern closely, however it does use its own logic in the implementation. Because the “C” is handled by the framework itself and most of the excitement in Django happens in models, templates and views, Django is often referred to as an *MTV framework*. In the MTV development pattern:

* **M stands for “Model,”** the data access layer. This layer contains anything and everything about the data: how to access it, how to validate it, which behaviors it has, and the relationships between the data. We will be looking closely at Django’s models in Chapter 4.
* **T stands for “Template,”** the presentation layer. This layer contains presentation-related decisions: how something should be displayed on a Web page or other type of document. We will explore Django’s templates in Chapter 3.
* **V stands for “View,”** the business logic layer. This layer contains the logic that accesses the model and defers to the appropriate template(s). You can think of it as the bridge between models and templates. We will be checking out Django’s views in the next chapter.

This is probably the only unfortunate bit of naming in Django, because Django’s view is more like the controller in MVC, and MVC’s view is actually a Template in Django. It is a little confusing at first, but as a programmer getting a job done, you really won’t care for long. It is only a problem for those of us who have to teach it. Oh, and to the flamers of course. (https://djangobook.com/model-view-controller-design-pattern/)

**R programming & Data Clustering:**

This part of the system architecture holds the core functionality with respect to data filtering and exploration. Calculation and manipulation related to data analysis and implementation of the unsupervised clustering concepts schemed within this segment.

**Data Exploration:**

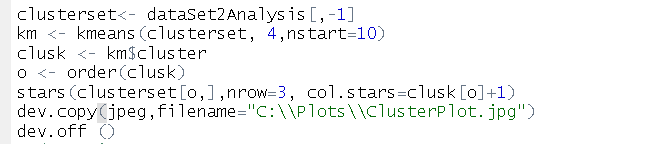
It is essential to list down what kind of information we are accumulating from the GPX files. Each of the GPX files consist of a series of nodes and each of the node holds four types of data. Timestamp, altitude, longitude & latitude. The combination of latitude and longitude pin points the location whereas the altitude states about the elevation details. The timestamp field provides information related to date and time. The starting node usually signifies the starting point of any journey. Whereas the last point usually lets us know about the end point of the journey. Having said that, we will consider the start and end points based upon time fields available within the file. Only these four fields are not good enough to find out the required clustering among different routes based upon the performed activity. We will make an attempt to derive some more fields from these original fields. The main objective to find out these derived fields is that it will provide us better understandability about the difficulty of a certain track. For an example, if there is a sharp inclination in the way then it is likely that a hiker or runner will face challenges and it will result in decrease of his/her speed. To figure out the change in elevation between successive nodes, we will plot the difference in ‘DeltaElev’ field. One interesting point is that the gadget being used, does not capture on regular intervals. And the speed of the person varies at different point of time as well. So, we would like to calculate the time differences between each two nodes in ‘TimeDiff’ field. From the combination of latitude and longitude fields, we get to realize the ‘Point’ field. From the ‘point’ field, we can compute the distance crossed between each successive node and the result is inserted in ‘GeoPointsDist’ field. One of the most significant things is to know how straight a path is. More bend found in a track, more likely it causes decrease in speed and less distance is covered as well. For activities like cycling, motor-cycling are very much impacted by the change of this field, ‘Angle’. For the start and end node we considered the angle as zero. To calculate the angle of any given point, we collected it’s prior and subsequent node for reference. If it is evident that all the three points are in a straight line, then also we consider its angle as zero.

Each of the result set is grouped with the respective filename and summary statistics is prepared for each of the files that are analysed during the process. Summary statistics provides us 6 parameters of data. They are min value, 1st quartile, median, mean, 3rd quartile, and maximum value. As our approach is to automatically form dataset so that it becomes suitable for clustering algorithm, we consider different approaches for different variables. It is to be specified that we will be performing clustering based on six fields. They are altitude, time, speed, timediff, DeltaElev, GeoPointDistance, Angle. We considered the median value for speed, timediff, and Angle field. But for altitude, DeltaElev, and time the different between maximum and minimum are evaluated. GeoPointDistance is measured as the summation of the total distance covered from the starting point to the end.

The relevant code snippet is attached below:

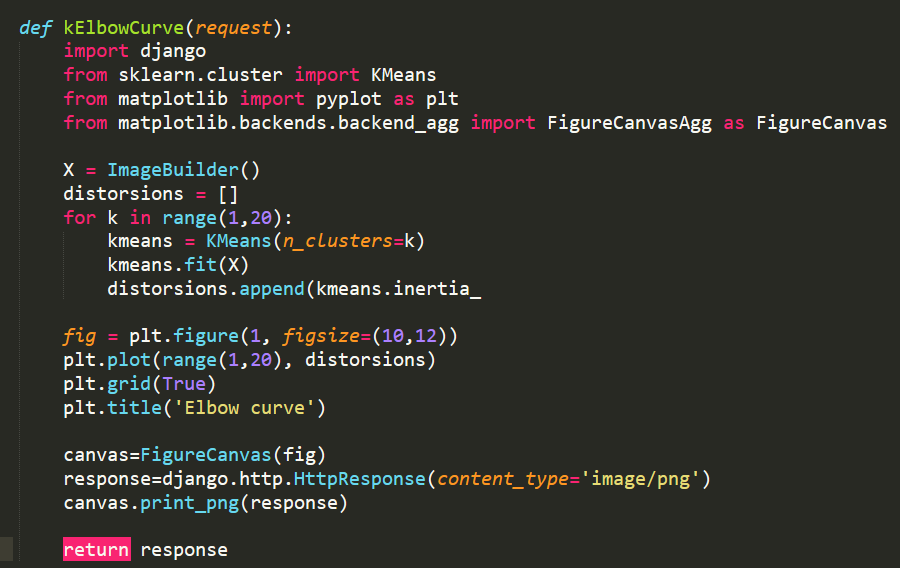


Based upon the mentioned fields value a new dataset is created, which will be used in different machine learning algorithm. In our project, we used KMeans algorithm to find out the similarity within different routes and form clusters with more closely related paths. The code snippet that implements the Kmeans algorithm is attached below:

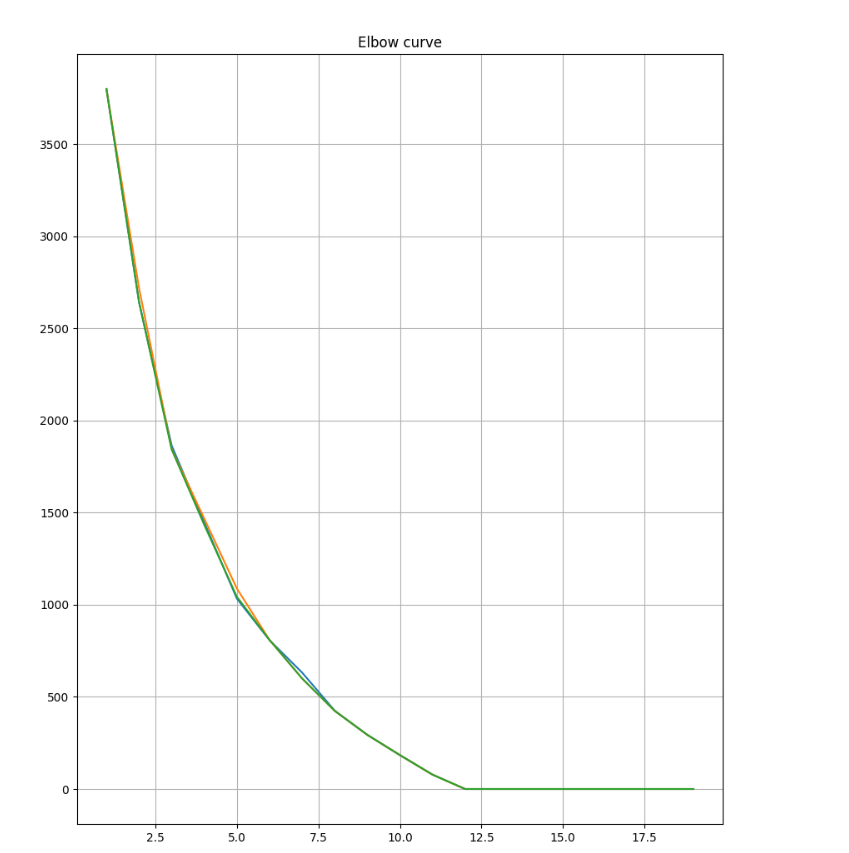


The output of the Kmeans is stored in a local directory as an image so that python web app can pull and make it visible in the user interface. From the Elbow curve we got to know about the optimum cluster number possible, which was 4 for the sample set of GPX file tested.

In Kmeans algorithm, it is very important to find out how many clusters we need to form based upon a given dataset. As per the unsupervised machine learning guideline, we created ‘Elbow’ curve in python programming to find out that significant number. The code snippet is given as:

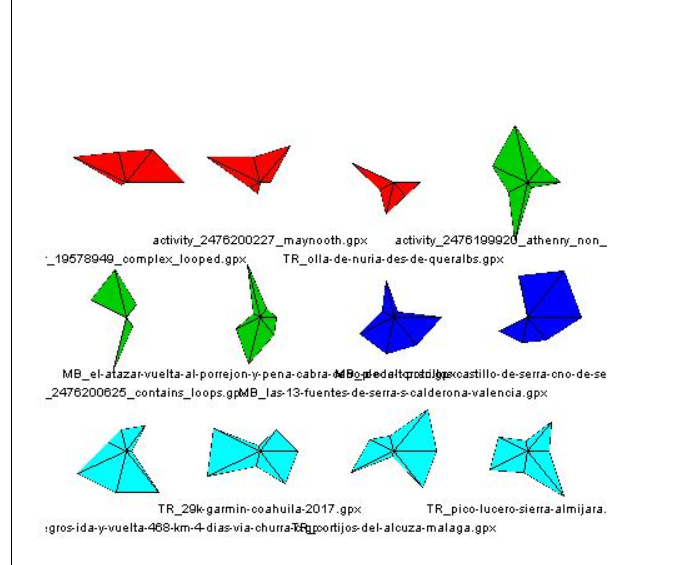


A sample output that got generated from the test dataset is given here:



As per the graph plotted, after 4th directional change (consider the movement towards the +ve x-axis) in the line, it became parallel to x-axis. So, in this instance maximum four clusters can be organized, not more than that.

In R code, when we run the Kmeans algorithm the generated output turned to be something similar like the one attached below:



Star diagrams are plotted from the multivariate dataset and each of the colour signifies the close match between given files. File names are mentioned below the stars and it is helpful to figure out which files are closely related from the output picture.

Clustering concepts and their implementation: Euclidean distance, Hierarchical clustering, Partitioning methods, Optimization

Leaflet

R(data load, extraction, derived fields, functions, Kmean and other algos from site)

Bootstrap

Local- Heroku AWS S3

**CHAPTER 5: Conclusions and Future Work.**

**Section 5:1 Summary of Thesis:**

**Section 5:2 Overall Evaluation**

**Section 5:3 Future Work**

**REFERENCES/BIBLIOGRAPHY**

**APPENDICES**

**Summarising academic papers**