



Subhajit Ghosh

17250762

Dissertation 2018

MSc in Data Science and Analytics

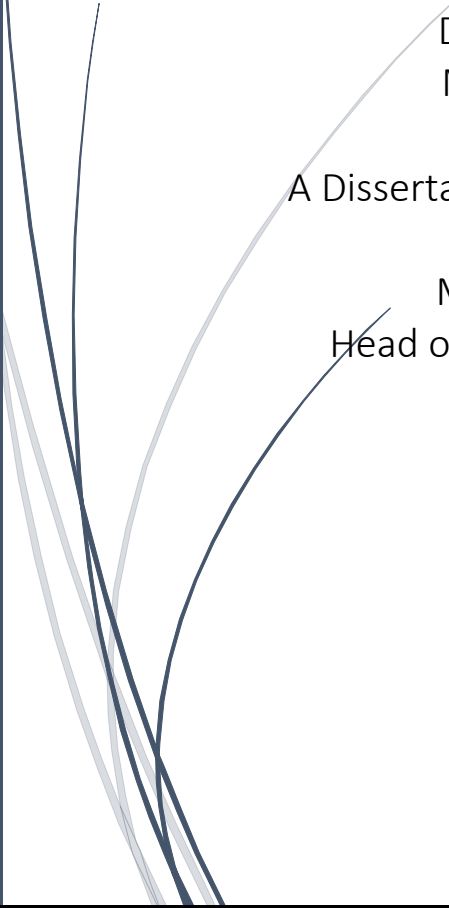


**Maynooth
University**

National University
of Ireland Maynooth

National University of Ireland, Maynooth

Maynooth, Co. Kildare, Ireland.



Department of Computer Science
Maynooth University, Maynooth,
Co. Kildare Ireland.

A Dissertation Submitted in partial fulfilment of the
requirement for the

MSc in Data Science and Analytics

Head of the Department: Dr Adam Winstanley

Supervisor: Dr. Peter Mooney

August 2018

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 Wikiloc 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 compute 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

The completion of this project has required an amalgamation of efforts and ideas from a lot of sources. In the end, I wish to acknowledge the intellectual input and support extended to me by all those directly or indirectly involved with this project.

I owe a profound sense of gratitude towards my supervisor Dr. Peter Mooney for giving me an opportunity to work under his guidance and for all the valuable guidance provided by him during the course of the project work. Without his informative suggestions and critical analysis, this project could not have been done. I thoroughly enjoyed those seasons where he helped me to resolve critical issues with simplest of approaches.

I would especially like to extend my heartfelt thankfulness towards my mentor for his constant encouragement and critical comments throughout the project. His contribution to this project extended beyond mere ideas up to a level of providing a congenial environment in the classroom. All through the duration of the project; his approachability made it remarkably easy for me to work and seek his inputs. I would also like to thank open source development platforms like Github, and question-answer site Stack-Overflow from the bottom of my heart. These platforms really helped me to learn new technologies and saved my life when I had to fix critical issues while in the development phase.

TABLE OF CONTENTS

TITLE	0
ABSTRACT	1
ACKNOWLEDGEMENTS	2
TABLE OF CONTENTS.....	3
LIST OF TABLES AND FIGURES.....	4
CHAPTER 1: Introduction	4
Section 1:1 Introduction to the problem	4
Section 1:2 Project Objectives	4
Section 1:3 Overall Solution.....	5
Section 1.4 Outline of the thesis.....	5
CHAPTER 2: Background and Related Work	6
Section 2:1 Related Literature and Other Works	6
Section 2:2 Outline of contributions.....	7
CHAPTER 3: Design and Solution Overview	9
CHAPTER 4: Analysis and Evaluation	20
CHAPTER 5: Conclusions and Future Work.....	25
Section 5:1 Summary of Thesis:.....	25
Section 5:2 Overall Evaluation	26
Section 5:3 Future Work	26
REFERENCES/BIBLIOGRAPHY	40
APPENDICES	27

LIST OF TABLES AND FIGURES

CHAPTER 1: Introduction

Based on different GPX files uploaded by different users around the globe, we are trying to club together different geo-tracks having same kind of features or characteristics. 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 plot associations 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 different characteristics, the project will be helpful to choose other tracks to increase the fitness level as per a selected sport activity.

Section 1:1 Introduction to the problem

There are lots of platforms available like strava.com, which display different kind of tracks (meant for cycling, hiking, running etc) in a geographic location. They simply allow the users to upload their GPX files generated from specific sports activity and show the trails as per coordinates. Now the problem is that if the user is keen to select only those routes which are having certain characteristics. For an instance tracks with no elevation changes (recommended for minor heart patients) or choosing tracks with nominal bends won't be possible in the aforesaid platforms.

Understanding the scope of improvement in the existing applications (eg: Runtastic, Endomondo, Strava), we tried to build an application, which can guide the users to opt for specific routes most relevant for their fitness related training or activity programs.

Section 1:2 Project Objectives

The project objectives are detailed below:

- 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.

- 2) Use machine learning approaches for the purposes of comparison and matching of the GPX files
- 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.
- 4) Allow the upload of GPX tracks from any geographical location and for any foot-based activity (hiking, jogging, walking or running).

Section 1:3 Overall Solution

To resolve the problems explained in section 1.1, we apprehended knowledge of database and machine learning, along with the expertise in software or application development. A web application is built using python programming and under the hood machine learning algorithms are executed to fetch the required outputs or plots.

Objective number 1, was met when we explored the GPX file structure and realized differently generated GPX files' uniformity (detailed in section 3 – data assemble and comprehension). Machine learning concepts were used (objective 2) to find out the parity among different GPX files characteristics (detailed in section 3 - implementation of ML Algorithm). From GPX files, generated from foot-based activities plots are generated in a web application, which secured our objective 3 & 4.

In this thesis, along with the fields available within GPX files (like Date time, longitude, latitude, elevation), we derived some more fields (like speed, Elevation change, angle change, total distance covered etc) by parsing the records found in any GPX file. The purpose of this data manipulation and derivation was to categorize the dataset in such a way so that same featured paths can be grouped together. In section 4, we compared outputs generated from k-means clustering and Hierarchical clustering algorithm on the basis of overall characteristics. We found that they formed groups with the same number of GPX files and with similar kind of characteristics. In the last part of section 4, we formed GPX tracks clusters only based on 'Angle'. We randomly chose a group and found that within that group, the shapes of the tracks from each file are identical.

Section 1.4 Outline of the thesis

Now that we get an overview of this paper, let's brief what we can expect in the remaining chapters.

- Chapter 2 details about the related work that is published or available in form of software or application. It also explains what new achievements are obtained during the development process
- Chapter 3 elaborately describes the actual work done through various diagrams and specifications of different models.

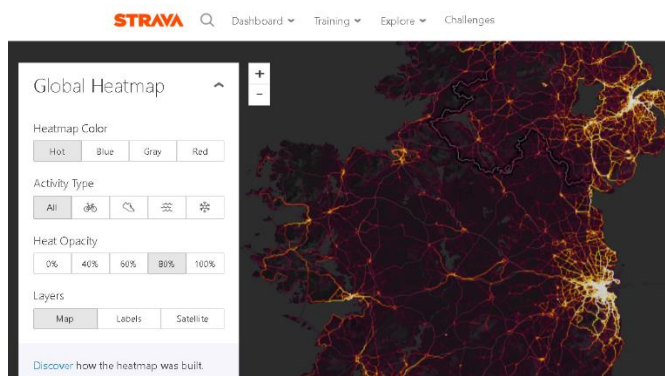
- Chapter 4 demonstrates mainly the obtained output from define solution in section 3. The generated output plots are cross verified by executing different algorithms at different phases. This chapter signifies how success this project work was.
- Chapter 5 summarizes all the work done and possesses a self-evaluation as well as thoughts about possible future work using the base provided by this thesis work.
- In appendices, a brief description is given to help the users to access the built application. The section is compressed with basic concepts and terminology definitions which were very crucial to build the overall product.

CHAPTER 2: Background and Related Work

Section 2:1 Related Literature and Other Works

This project is inspired by the academic paper, 'Identifying the sports Activity of GPX Tracks'. It was demonstrated in the paper how 8500 GPS tracks from 10 different kinds of sports were analyzed and how accurately a suitable sports activity for a certain track was anticipated. Nowadays, 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 on those tracks. Actually, these apps give the user an option to upload GPX files, which got generated from a sports activity.

A basic Strava page looks like:

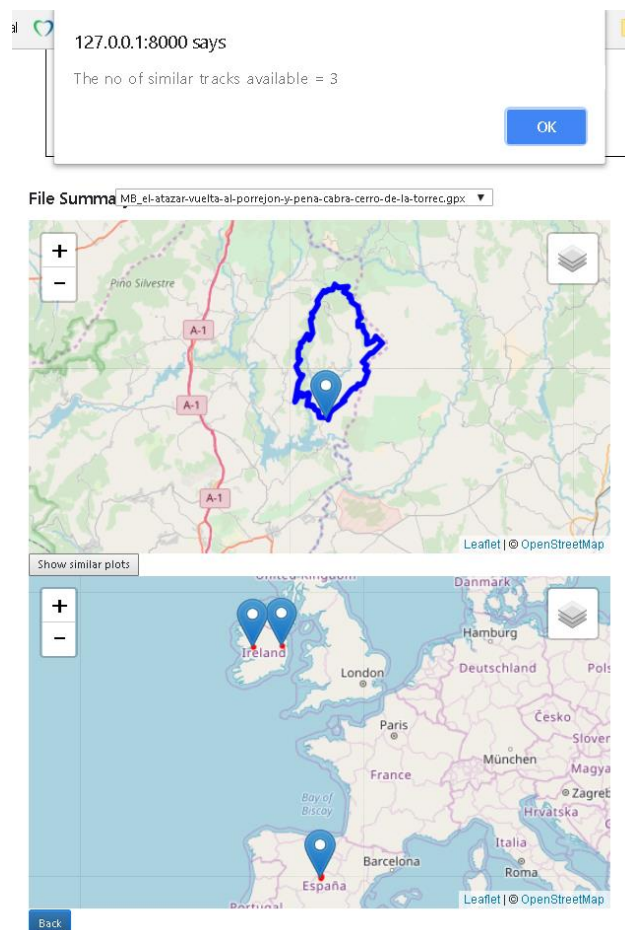


It is witnessed that map is populated as per uploaded GPX tracks.

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 a selection between similar tracks. Along with other clustering algorithms we used K-means algorithm and generated as per characteristics like altitude, total-time-taken, speed, elevation-difference, total-distance-covered, avg-angle-change. While considering the overall response, all the mentioned characteristics are given the same weightage. But we can provide users a 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 to any user's fitness program.

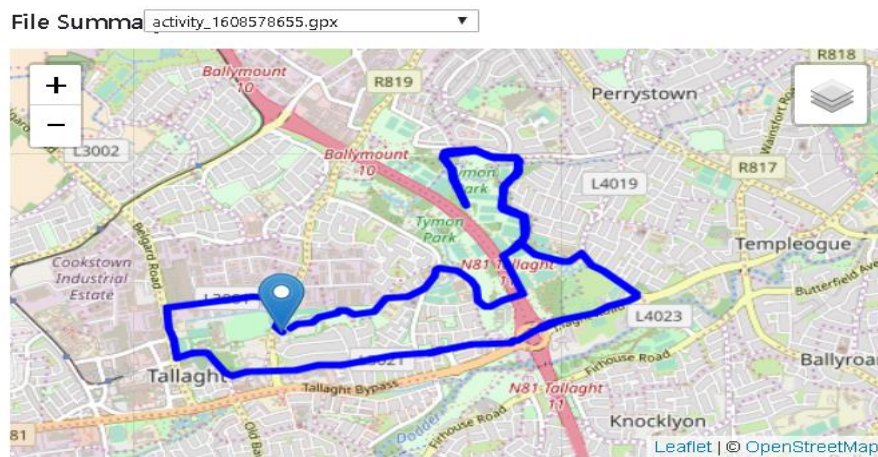


Though we are maintaining basic UI features in the project, it is enough to depict the outcome achieved. The first map was populated as per a track selection by a user and the second map detailed all the tracks which are having similar characteristics. For this instance, only 3 similar tracks are found and prompted in an alert box.

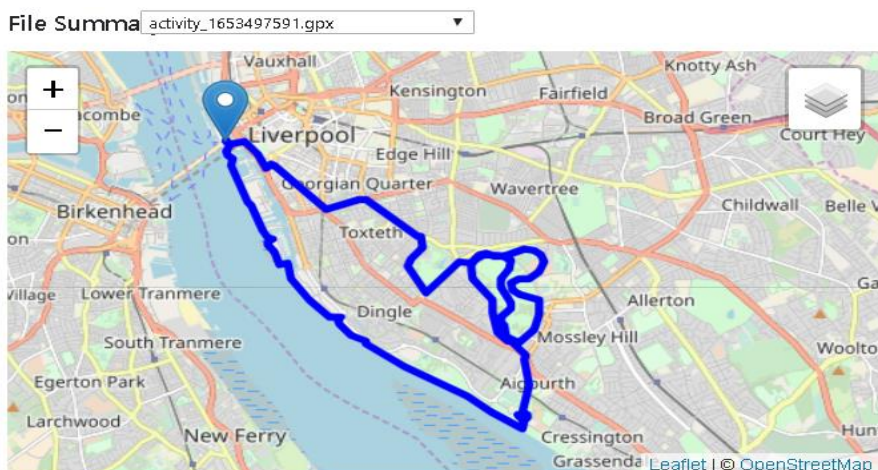
Please note tracks grouped in the same cluster might not have a similar shape as the clustering algorithm is performed on a group of characteristics.

If user imposes rule for only 'change in angle of the route', clustered trails will have similar kinds of shape rather than anything else. As explained in chapter 4, we grouped test files through hierarchical clustering and found that the clubbed files shapes are very similar to each other. As in this application, the user will have the option to set up the rule based upon which clustering will take place and characteristic defined within the rule will form the grouping of the files. The shapes of 2 files found in the same group look like:

2nd files(activity_1608578655.gpx) shape:



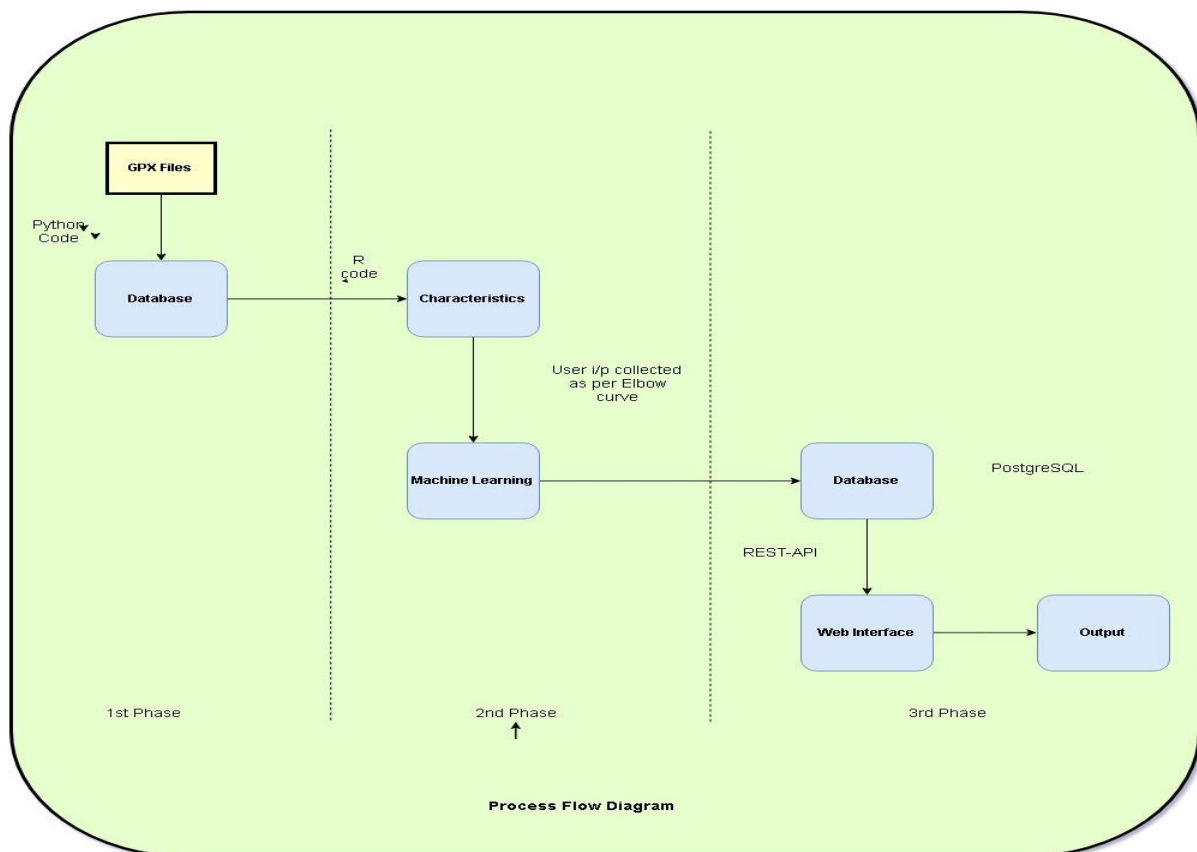
3rd files(activity_1653497591.gpx) shape:



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 was 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 the 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 the 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 parts has its own significance and the roles of them are detailed within their own divisions. We divided the execution flow into 3 main phases. Each of them is detailed below.



➤ 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](#) designed as a common [GPS](#) data format for software applications. ([wikipedia, n.d.](#))). 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 through 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:

```
<trkseg>
  <trkpt lat="53.1422794796526432037353515625" lon="-7.05871456302702426910400390625">
    <ele>65.59999847412109375</ele>
    <time>2017-08-21T11:37:24.000Z</time>
    <extensions>
      <ns3:TrackPointExtension>
        <ns3:cad>0</ns3:cad>
      </ns3:TrackPointExtension>
    </extensions>
  </trkpt>
  <trkpt lat="53.14214612357318401396669921875" lon="-7.05810385756194591522216796875">
    <ele>65.40000152587890625</ele>
    <time>2017-08-21T11:37:34.000Z</time>
    <extensions>
      <ns3:TrackPointExtension>
        <ns3:cad>85</ns3:cad>
      </ns3:TrackPointExtension>
    </extensions>
  </trkpt>
```

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 parsing and fetched information like: Filename, Longitude, Latitude, Altitude, Time & Speed. The relevant code-snippet is found below.

```
137
138 def loadData(path, gpxFileName):
139     gpx = gpxpy.parse(open(path + gpxFileName))
140
141     print("{} track(s)".format(len(gpx.tracks)))
142     track = gpx.tracks[0]
143     print("{} segment(s)".format(len(track.segments)))
144     segment = track.segments[0]
145
146     print("{} point(s)".format(len(segment.points)))
147
148     data = []
149     segment_length = segment.length_3d()
150     for point_idx, point in enumerate(segment.points):
151         data.append([gpxFileName, point.longitude, point.latitude,
152                     point.elevation, point.time, segment.get_speed(point_idx)])
153
154     columns = ['FileName', 'Longitude', 'Latitude', 'Altitude', 'Time', 'Speed']
155     df = DataFrame(data, columns=columns)
156     df.head()
157     print(df.head())
158
159     return df.values
```

All 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 <> "";`

```
40 select * from gpxcontenttable;
```

	id	filename	longitude	latitude	altitude	time	speed	point
	integer	character varying (500)	character varying (500)	character varying (500)	character varying (500)	timestamp without time zone	numeric	geometry
1	1	activity_19578949_com...	-7.058714563027024	53.14227947965264	65.5999984741211	2017-08-21 11:37:24	7.004791731067535	0101000...
2	2	activity_19578949_com...	-7.05841826274991	53.14221485517919	65.5999984741211	2017-08-21 11:37:27	5.095641077154511	0101000020E6100000000A0AB1F3C1CC00000CB3636924A40

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

The 2nd phase mainly completed within the 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 the 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 intention of creating new derived fields are explained in grid below.

Field Name	Purpose of Creation
TimeDiff	Time gap between 2 successive waypoints
DistanceCovered	distance derived from the speed within each TimeDiff
DeltaElev	How altitude varied between nodes
GeoPointsDist	Actual distance between 2 points
Angle	signifies how much a user needs to turn from the straight line

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*)

```
{ #----calculate the distance betw 2 Geoms in R-----
#install.packages("geosphere")
#install.packages("purrr")
get_geo_distance = function(long1, lat1, long2, lat2, units = "miles") {
  loadNamespace("purrr")
  loadNamespace("geosphere")
  longlat1 = purrr::map2(long1, lat1, function(x,y) c(x,y))
  longlat2 = purrr::map2(long2, lat2, function(x,y) c(x,y))
  distance_list = purrr::map2(longlat1, longlat2, function(x,y) geosphere::distHaversine(x, y))
  ##distance_m = list_extract(distance_list, position = 1)
  distance_m = distance_list[1]
  if (units == "km") {
    distance = distance_m[[1]] / 1000.0;
  }
  else if (units == "miles") {
    distance = distance_m[[1]] / 1609.344
  }
  else {
    distance = distance_m[[1]]
    # This will return in meter as same way as distHaversine function.
  }
  distance
}
```

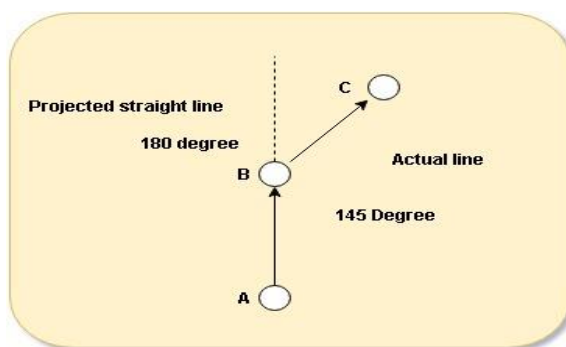
Get_geo_distance function is called each time to determine the distance between every consecutive way-points. DistanceCalc function was written to populate GeoPointsDist field. The start and end point will always have zero in their GeoPointDist field.

```
DistanceCalc<- function(x){
  x <- x[order(x$filename, x$time),]

  vect=0
  filenameTemp=x$filename[1]
  for(i in 1:length(x$DateTime))
  {
    if(i==1) {next()}
    else if(filenameTemp==x$filename[i]){
      vect<- c(vect,get_geo_distance(x$longitude[i],x$latitude[i],x$longitude[i-1], x$latitude[i-1], units = "km"))
    }
    else{
      vect<- c(vect,0)
    }
    filenameTemp=x$filename[i]
  }
  vect
}

gpxcontent$GeoPointsDist<- DistanceCalc(gpxcontent)
"
```

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, especially in motor related vehicles. Though in this thesis paper we are focused to test data generated from running/walking activities, the angle has a significant role in making the overall clustering success.



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 degrees. Cos 180 is equal to -1. C is 45 degrees 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:

```

AngleMeasure<- function(x){
  x <- x[order(x$filename, x$time),]

  vect=0
  filenameTemp=x$filename[1]
  for(i in 1:length(x$id))
  {
    if(i==1) {next()}
    else if(i==length(x$id)){vect<- c(vect,0)}
    else if(filenameTemp==x$filename[i]){
      vect<- c(vect,(180 - acos(((get_geo_distance(gpxcontent$longitude[i-1],gpxcontent$latitude[i-1],gpxcontent$longitude[i+1],gpxcontent$latitude[i+1]))^2
      -(gpxcontent$GeoPointsDist[i])^2)-(gpxcontent$GeoPointsDist[i+1])^2)
      /(-2*(gpxcontent$GeoPointsDist[i]*gpxcontent$GeoPointsDist[i+1])))*180/pi))
    }
    else{
      vect<- c(vect,0)
    }
    filenameTemp=x$filename[i]
  }
  vect
}

gpxcontent$Angle<- AngleMeasure(gpxcontent)

```

To 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.

```

> statSet<- data.frame(gpxcontent[gpxcontent$filename==unique(gpxcontent$filename)[1],c(1,3:7,9:14)])
> summary(statSet)

```

id	longitude	latitude	altitude	time	speed	DateTime	TimeDiff
Min. : 1.0	Min. :-7.083	Min. :53.13	Min. :60.40	Min. :2017-08-21 11:37:24	Min. :0.7656	Min. :2017-08-21 10:37:24	Min. : 0.000
1st Qu.:170.8	1st Qu.: -7.070	1st Qu.:53.14	1st Qu.:63.00	1st Qu.:2017-08-21 11:54:12	1st Qu.:3.4108	1st Qu.:2017-08-21 10:54:12	1st Qu.: 6.000
Median :340.5	Median :-7.063	Median :53.14	Median :64.60	Median :2017-08-21 12:12:26	Median :3.5597	Median :2017-08-21 11:12:26	Median : 6.000
Mean :340.5	Mean :-7.061	Mean :53.14	Mean :64.21	Mean :2017-08-21 12:12:03	Mean :3.4646	Mean :2017-08-21 11:12:03	Mean : 6.088
3rd Qu.:510.2	3rd Qu.: -7.055	3rd Qu.:53.15	3rd Qu.:65.40	3rd Qu.:2017-08-21 12:29:18	3rd Qu.:3.6768	3rd Qu.:2017-08-21 11:29:18	3rd Qu.: 6.000
Max. :680.0	Max. :-7.032	Max. :53.15	Max. :67.60	Max. :2017-08-21 12:46:24	Max. :7.0048	Max. :2017-08-21 11:46:24	Max. :51.000
DistanceCovered	DeltaElev	GeoPointsDist	Angle				
Min. :0.000000	Min. :-1.0000000	Min. :0.00000	Min. : 0.0				
1st Qu.:0.005777	1st Qu.: -0.1999969	1st Qu.:0.02077	1st Qu.:103.2				
Median :0.005979	Median : 0.0000000	Median :0.02160	Median :103.3				
Mean :0.005763	Mean :-0.0002941	Mean :0.02058	Mean :100.5				
3rd Qu.:0.006174	3rd Qu.: 0.1999969	3rd Qu.:0.02226	3rd Qu.:103.5				
Max. :0.027877	Max. : 0.7999992	Max. :0.05099	Max. :170.6				

After analysis and discussion, we followed different conventions for different attributed. For speed, angle, TimeDiff we considered the 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 fields 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 the different list.

Name	Type	Value
summaryList	list [25]	List of length 25
[[1]]	list [1]	List of length 1
[[1]]	list [6 × 13] (S3: data.frame)	A data.frame with 6 rows and 13 columns
id_gist	character [6]	'Min.: 1.0 '1st Qu.: 589.8 'Median: 1178.5 'Mean: 1178.5 '3rd ...
longitude	character [6]	'Min.: -6.014 '1st Qu.: -5.997 'Median: -5.985 'Mean: -5.985 '3rd ...
latitude	character [6]	'Min.: 37.36 '1st Qu.: 37.38 'Median: 37.39 'Mean: 37.39 '3rd Qu.: ...
altitude	character [6]	'Min.: 1.20 '1st Qu.: 10.40 'Median: 12.80 'Mean: 12.97 '3rd Qu.: ...
time	character [6]	'Min.: 2017-02-19 07:30:22 '1st Qu.: 2017-02-19 08:09:38 'Median: 2017-02 ...
speed	character [6]	'Min.: 1.620 '1st Qu.: 4.347 'Median: 4.481 'Mean: 4.519 '3rd ...
DateTime	character [6]	'Min.: 2017-02-19 07:30:22 '1st Qu.: 2017-02-19 08:09:38 'Median: 2017-02 ...
TimeDiff	character [6]	'Min.: 0.000 '1st Qu.: 3.000 'Median: 5.000 'Mean: 4.025 '3rd Qu.: ...
DistanceCovered	character [6]	'Min.: 0.000000 '1st Qu.: 0.003787 'Median: 0.005963 'Mean: 0.005036 ...
DeltaElev	character [6]	'Min.: -4.60000 '1st Qu.: -0.20000 'Median: 0.00000 'Mean: 0.00348 ...
GeoPointsDist	character [6]	'Min.: 0.00000 '1st Qu.: 0.01365 'Median: 0.02141 'Mean: 0.01813 ...
Angle	character [6]	'Min.: 0.00 '1st Qu.: 103.21 'Median: 103.67 'Mean: 96.97 '3rd ...
filename	character [6]	'activity_1580981561.gpx' 'activity_1580981561.gpx' 'activity_1580981561.gpx' 'a ...
[[2]]	list [1]	List of length 1
[[3]]	list [1]	List of length 1

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':

```
> summaryList[[1]][[1]][["speed"]]
[1] "Min. :0.7656 "1st Qu.:3.4108 "Median :3.5597 "Mean :3.4646 "3rd Qu.:3.6768 "
[6] "Max. :7.0048 "
```

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

```
> as.numeric(strsplit(summaryList[[1]][[1]][["speed"]][3],":")[1][2])
[1] 3.5597
```

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

```
dataSet2Analysis<- c()
library(lubridate)
populate_dataset2Analysis<- function (fname){
  j<- 0
  for(i in 1:length(summaryList)){
    if(summaryList[[i]][[1]][["filename"]][1]==fname){
      j=i
      break
    }
  }
  v<- ymd_hms(trimws(strsplit(summaryList[[j]][[1]][["time"]][1],":")[1][2],which='b'),tz=Sys.timezone())
  v1<- ymd_hms(trimws(strsplit(summaryList[[j]][[1]][["time"]][6],":")[1][2],which='b'),tz=Sys.timezone())
  dataR<- c(fname,
    #altitude
    as.numeric(strsplit(summaryList[[j]][[1]][["altitude"]][6],":")[1][2]),
    - as.numeric(strsplit(summaryList[[j]][[1]][["altitude"]][1],":")[1][2]),
    #time
    difftime(v,v1, units = "secs"),
    #speed
    as.numeric(strsplit(summaryList[[j]][[1]][["speed"]][3],":")[1][2]),
    #timediff
    as.numeric(strsplit(summaryList[[j]][[1]][["TimeDiff"]][3],":")[1][2]),
    #DeltaElev
    as.numeric(strsplit(summaryList[[j]][[1]][["DeltaElev"]][6],":")[1][2]),
    - as.numeric(strsplit(summaryList[[j]][[1]][["DeltaElev"]][1],":")[1][2]),
    #GeoPointDistance
    sum(gpxcontent[gpxcontent$filename==fname,$GeoPointsDist]),
    #Angle
    as.numeric(strsplit(summaryList[[j]][[1]][["Angle"]][3],":")[1][2])
  )
  dataR
}
for(i in 1:length(unique(gpxcontent$filename))){
  dataSet2Analysis<- rbind(dataSet2Analysis,populate_dataset2Analysis(unique(gpxcontent$filename)[i]))
}
```

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

Untitled1*	R-PostGresql-Bridge.R	summaryList	dataSet2Analysis	Clustering-Code-Final.R	test.R			
Filter								
filename	altitude	time	speed	timeDiff	DeltaElev	totalDist	Angle	
activity_19578949_complex_looped.gpx	7.2	4140	3.5597	6	1.7999992	13.9970697406647	103.3	
activity_2476199920_athenry_non_loop.gpx	3.4	2016	4.9711	4	3	9.99606886798002	103.8	
activity_2476200227_maynooth.gpx	13.2	4785	3.5579	6	1.8	16.4300179978495	103.3	
activity_2476200625_contains_loops.gpx	-1.8	1666	4.831	4	1.600002	8.017681288932	103.9	
MB_el-atazar-vuelta-al-porreon-y-pena-cabra-cerro-de-la-to...	1593.3	14859	4.85123	17	30.287	50.8970640758572	111.2	
MB_las-13-fuentes-de-serra-s-calderona-valencia.gpx	640.6	12705	4.064	19	34.185	41.1919688265815	112.3	
MB_p-coeli-potrillos-castillo-de-serra-cno-de-serra-a-gatova...	652.2	8899	4.3893	16	31.169	28.8113634347491	111.8	
MB_valencia-ojos-negros-ida-y-vuelta-468-km-4-dias-via-ch...	1196.31	216740	2.5346	300	71.126	455.666740052818	113.1	
TR_29k-garmin-coahuila-2017.gpx	3424.6	20403	1.4801	50	74.706	27.4012593138775	112.52	
TR_cor tijos-del-alcuza-malaga.gpx	560.28	8084	2.3019	33	48.582	14.0349765544569	113	
TR_olla-de-nuria-des-de-queralbs.gpx	2843.6	1136879	1.029192	46	1423.386	388.188994845428	108.9	
TR_pico-lucero-sierra-almijara.gpx	1763.4	25606	1.7473	36	48.373	28.5172457509234	115.1	

Now we have the dataset ready for various clustering algorithm to run. We mainly used a) Hierarchical clustering & b) Kmeans algorithm in our application. (stanford.edu, n.d.)

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 $x_{i1}, x_{i2}, \dots, x_{ip}$, and case j with coordinates $x_{j1}, x_{j2}, \dots, x_{jp}$.

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

$$d(i, j) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \dots (x_{ip} - x_{jp})^2}$$

$$= \sqrt{\sum_{k=1}^p (x_{ik} - x_{jk})^2}$$

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

$$d(i, j) = \sum_{k=1}^p (x_{ik} - x_{jk})^2$$

- iii. The Manhattan distance between case i and case j is:

$$d(i, j) = \sum_{k=1}^p |x_{ik} - x_{jk}|$$

For further calculation in regarding cluster, we considered the **Euclidean** method for finding out the 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 $U = u_1, u_2, \dots$ and $V = v_1, v_2, \dots$ 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 .

$$d(U, V) = \min_{u \in U, v \in V} d(u, v)$$

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

$$d(U, V) = \max_{u \in U, v \in V} d(u, v)$$

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

$$d(U, V) = \frac{\sum_{u \in U, v \in V} d(u, v)}{rs}$$

- **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 .

$$d(U, V) = d(\mathbf{u}, \mathbf{v})$$

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

1. Data assignment 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 c_i is the collection of centroids in set C , then each data point x is assigned to a cluster based on

$$\operatorname{argmin}_{c_i \in C} \operatorname{dist}(c_i, x)^2$$

where $\operatorname{dist}(\cdot)$ is the standard (L_2) Euclidean distance. Let the set of data point assignments for each i^{th} cluster centroid be S_i .

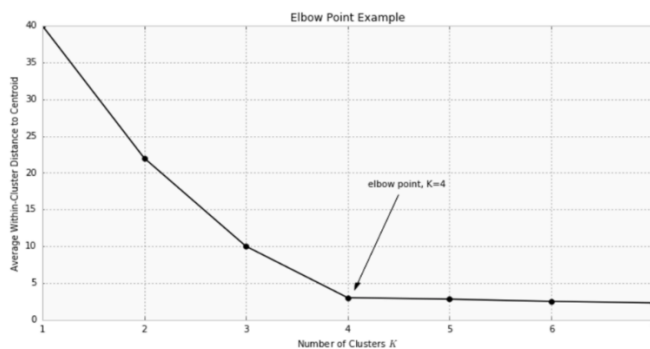
2. Centroid update step:

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

$$c_i = \frac{1}{|S_i|} \sum_{x_i \in S_i} x_i$$

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 .

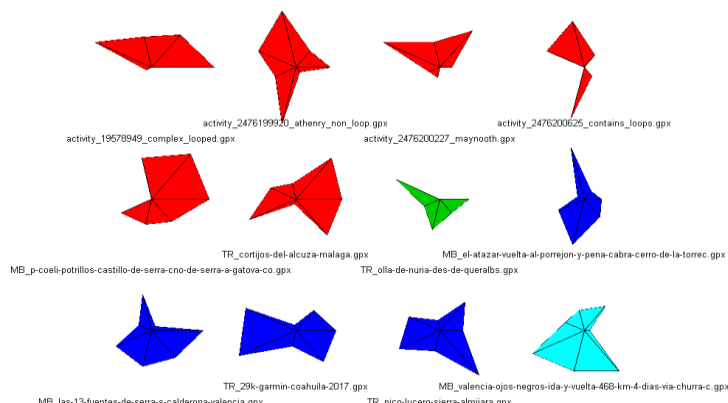


After 4th point the curve became almost parallel to the x-axis. That means for further grouping of the dataset will make the TWSS tend to zero, which is not required. So, making 4 clusters from the figure mentioned above would be most relevant.

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

```
clusterset<- DataSet2Analysis[,-1]
km <- kmeans(clusterset, 4,nstart=10)
clusk <- km$cluster
o <- order(clusk)
stars(clusterset[o,],nrow=3, col.stars=clusk[o]+1, col.segments = c(2,3,4,5))
dev.copy(jpeg,filename="C:\\subhajit\\project\\smartblog\\smartblogproject\\Plots\\ClusterPlot.jpg")
dev.off()
```

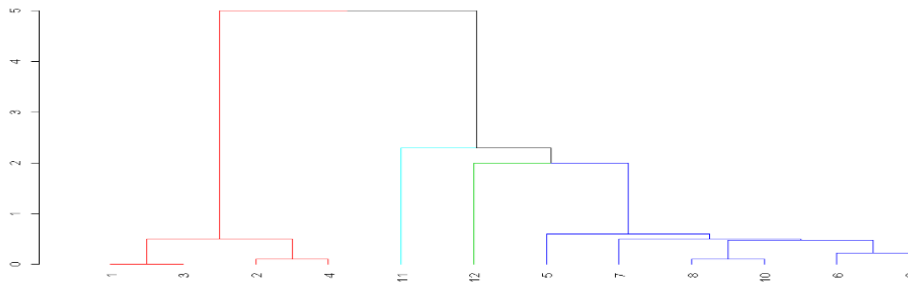
The clustered output was generated in star-like images and pulled in webpages at later point of time.



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

```
#install.packages("dendextend")
d<- dist(clusterset[,7],"euclidean")
h<- hclust(d,"single")
d1 <- as.dendrogram(h)
library(dendextend)
d2=color_branches(d1,k=4, col=c(2,5,3,4)) # auto-coloring 4 clusters of branches.
plot(d2)
```

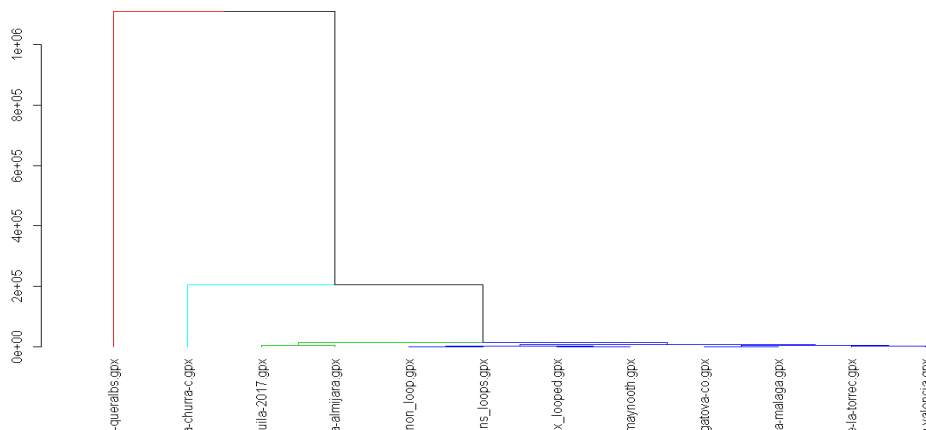
The generated output was:



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

```
d<- dist(clusterset,"euclidean")
h<- hclust(d,"average")
d1 <- as.dendrogram(h)
d2=color_branches(d1,k=4,col=c(2,5,3,4)) # auto-coloring 4 clusters of branches.
plot(d2)
```

And the generated output was like:

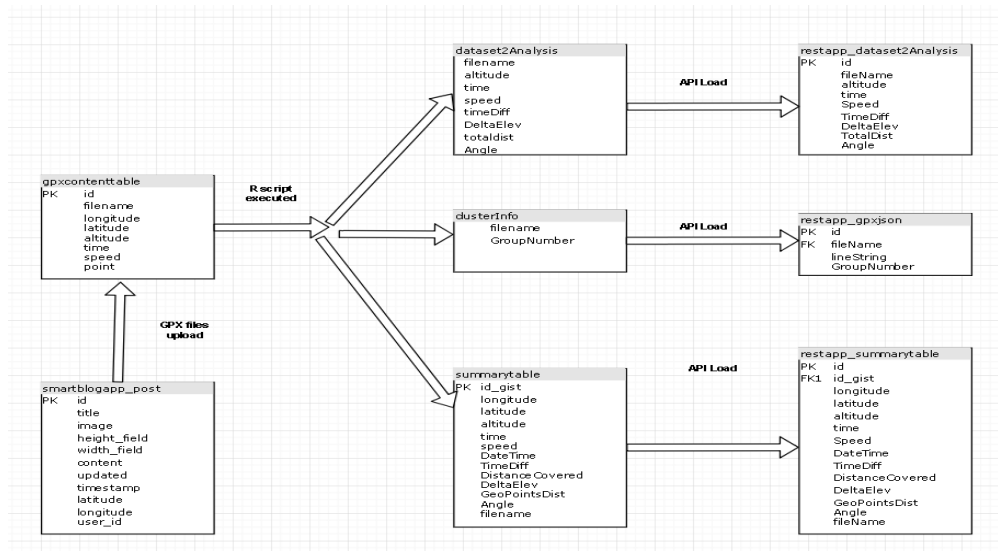


➤ **Analysed Data Store & Output Visualization in Web platform [3rd Phase]:**

The data science related main calculation and manipulation were done in 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 the `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`. Thereafter the 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:

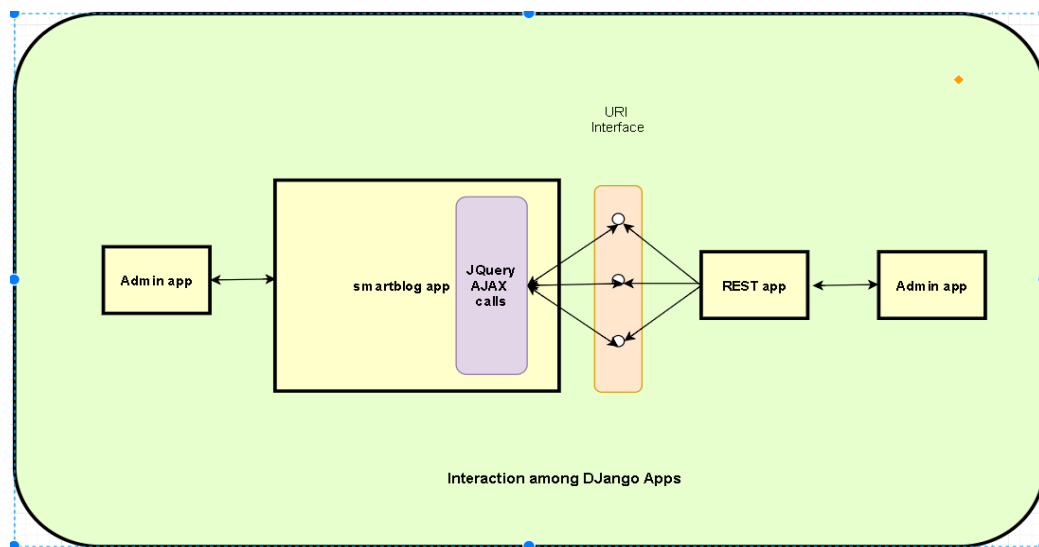
```

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 the `gpxcontenttable` table to form a linestring and then populated `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.

To understand how the configured REST-API and other app modules are interacting among each other, we will try to explain with a diagram.

Interaction among Apps built using Django:



The Django project compresses 3 apps within itself. Admin, Smartblog & Rest apps make the interaction among themselves to provide the overall functionalities. Both Smartblog and Rest apps are customizable from the Admin app. The Rest app creates a 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 the data model section. The smartblog app is built with a couple of templates (rendered as HTML pages). Within each templates, javascript/Jquery code is written to consume rest-response as per requirement and reflect that in the UI. (djangoproject.com, n.d.)

CHAPTER 4: Analysis and Evaluation

35 GPX track taken from running/jogging activities in Ireland, UK, Spain, Switzerland, and Sweden. We purposely chose 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. We had objectives for this project (discussed in section 1.2) and we successfully met all the requirements.

The GPX files generated in different geographic location and by different people does not make any changes in our evaluation process, as each of the GPX files is built following uniform schema. GPX files structure and element detail are described thoroughly in section 3. Each of the files is having an element:

`xsi:schemaLocation=`<http://www.topografix.com/GPX/1/1> <http://www.topografix.com/GPX/11.xsd>

The .xsd defines the schema, which provides uniformity. It gives us the opportunity to proceed with data validation, derivation, and manipulation. After that, we applied various machine learning concepts to find the similarities present among test files generated by different means. The ML concepts and how they were applied in the application were discussed in section 3. Now, will try to evaluate what were the outcome of our analysis over 36 test files.

As discussed in section 3, for each of execution process we need to build a summary table, which accumulates the required values of the fields (altitude, time, speed, timeDiff, DeltaElevation, Total Distance & Angle). It is shown in the web application (details page) as:

C:\gpxFiles\

Upload GPX Files

Load Summary API

Files Summary

Elbow Curve

K-means Clusters

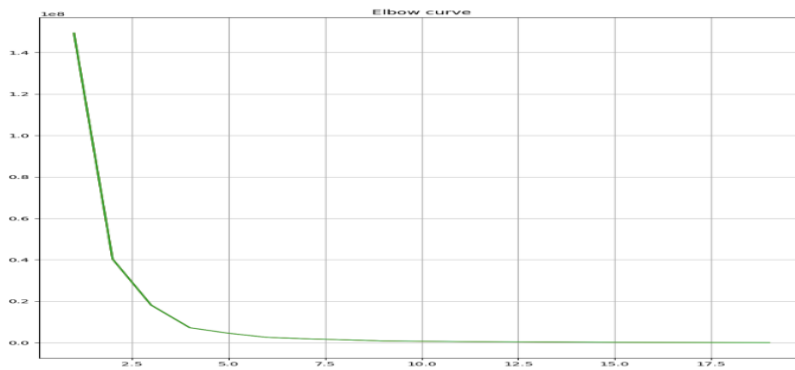
Dendrogram

Summary Table

File Name	Altitude	Time	Speed	TimeDiff	Delta Elevation	Total Distance	Angle
activity_1580981561.gpx	24.2	9482	4.481	5	7.2	42.7250875653998	103.67
activity_1608578655.gpx	101.4	3762	4.0439	4	5.8000046	13.7413622293525	105.14
activity_1653497591.gpx	51	4457	4.709	4	20.6	20.9593408541155	105.04
activity_1666612514.gpx	190.4	6767	3.865	5	7.8	25.7619808565679	103.6
activity_2489323352.gpx	96.2	5599	3.56571	6	6.2	17.1886858402845	103.49
activity_2497357501.gpx	67.8	2507	3.9636	5	5.400003	9.14064459849717	103.4
activity_2499685769.gpx	68	4389	3.7314	6	7.2	16.1021376064661	103.4
activity_2568998845.gpx	58.4	2313	3.5921	6	7.200002	8.04867864130036	103.4
activity_2595492741.gpx	82.6	8874	3.6121	6	6.399995	30.6107625338199	103.3
activity_2643148359.gpx	47.4	10006	4.3773	5	8.4	42.567637811863	103.4
activity_2650293777.gpx	71.6	3591	3.667	6	5.6	12.8675760603364	103.3
activity_2705776099.gpx	87.6	2694	3.76	6	6.4	9.62696877436882	103.4
activity_2710130093.gpx	74.6	3591	3.72169	6	5.8	10.0646752842947	103.29
activity_2743178994.gpx	82.2	3449	3.928	6	5.6	12.9643952693961	103.3
activity_2743179064.gpx	72.8	2329	3.8327	6	5.800005	8.08728653767363	103.31
activity_2743179162.gpx	74.6	2428	3.884	6	5.6	8.84460821323519	103.3
activity_2777997709.gpx	80	2548	4.411	5	5.4	11.2460877413222	103.26
activity_2824950282.gpx	74.4	3669	3.6005	6	5.8000046	12.8738696312777	103.3
activity_2834664660.gpx	216.8	5924	3.565	6	7.6	20.0065256879196	103.4
activity_2841601898.gpx	86	3464	3.6	6	8.000002	12.1470917777586	103.6
activity_2842747188.gpx	137.6	3836	3.805	6	6.20001	14.4670576451081	103.4
activity_2844248629.gpx	90.8	2343	3.468	6	5.4	8.05159798212112	103.3
activity_2845684486.gpx	66.2	3772	3.926	5	5.6	13.8316639373153	103.4
activity_2848000976.gpx	63	3044	3.4519	6	5.400003	10.3778178841844	103.4
activity_2848149140.gpx	131.2	4396	3.787	6	6.399995	16.1064688621847	103.3
activity_2850717823.gpx	74.4	2100	3.715	6	5.800005	7.68017982500786	103.3
activity_2850717933.gpx	443.8	2496	3.3594	6	7.6	6.56725363088418	103.51
activity_2850718186.very_complex.gpx	45.8	3289	4.6709	4	5.8	12.0387104442812	104.75
activity_2850718220.gpx	61.8	3141	3.6382	6	6.0000015	11.3303281168541	103.34
activity_2850718767.gpx	84.8	4176	3.58503	6	5.8000046	12.4669948136546	103.3
activity_2850718984.gpx	74.6	2638	3.906	6	5.7999969	10.0027538537346	103.4

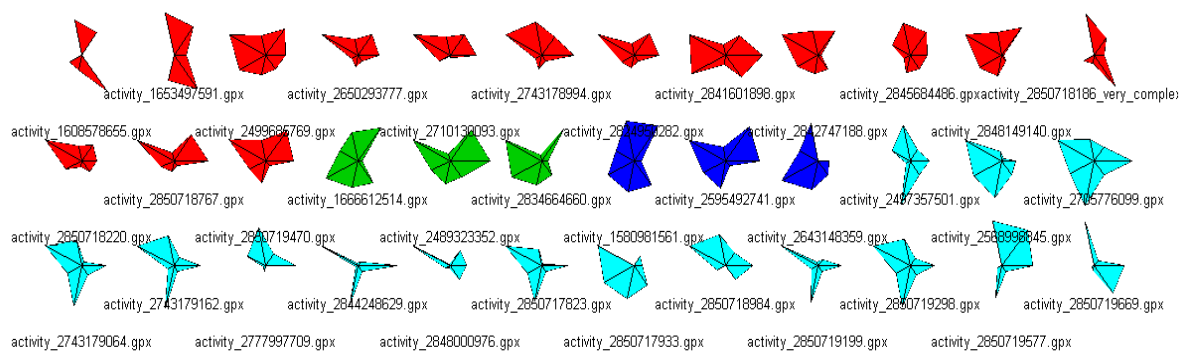
Please be noted that each of the field values might denote different units (like for Time, it is seconds. Whereas for Angle, it is degree), but each of the 7 fields is holding the same weightage. The ultimate clustering will take place from the overall impact of all the 7 fields. We also set rules to group result set as per individual field. But for the time being let's analyse the overall impact.

Prior to the execution of any unsupervised machine learning clustering algorithm, it is essential to predict how many clusters there should be for any given set of data. For that purpose, we plot 'Elbow curve' and show it in the webpage. The figured-out curve was like:



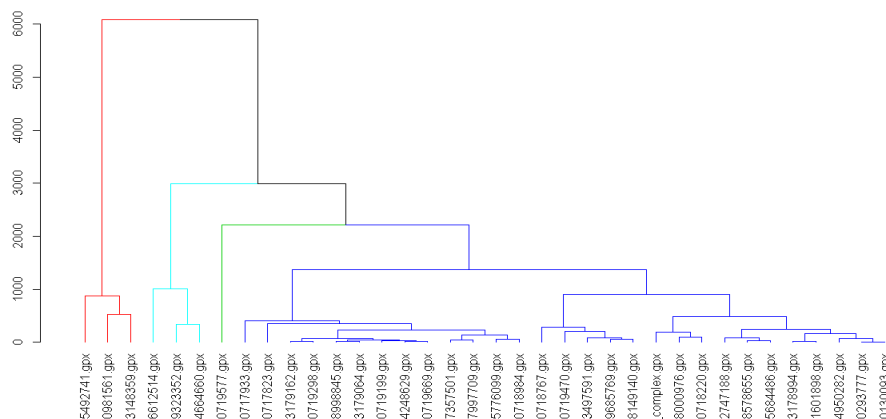
From the diagram it was clear that after 4th point of directional change, the curve became almost parallel to the x-axis. Note: we are moving towards +ve x-axis and the starting point is also considered in point calculation. So, it would be better if we divide the whole dataset on 4 different groups and run the clustering algorithms.

For K-means clustering, we achieved the result as: (Star plot)



Here, 2 groups are containing 3 files each whereas the other 2 groups are consisting with 12 files each.

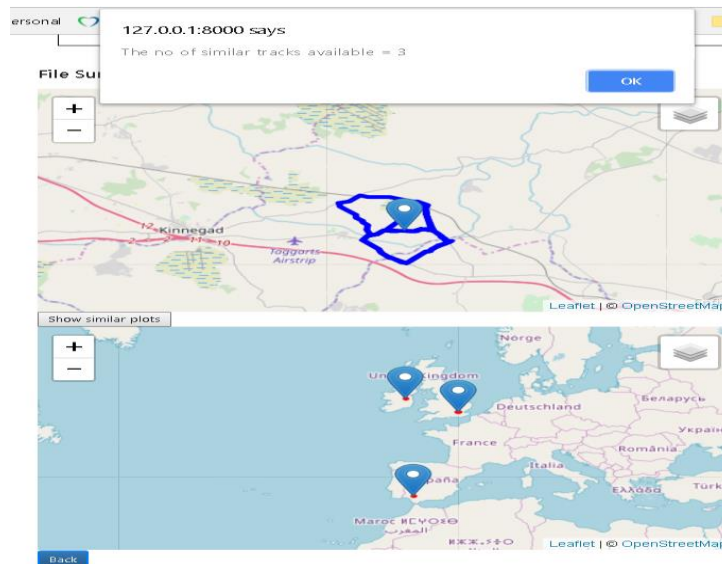
When we tried to find out result using Hierarchical clustering (**Average linkage distance**), the achieved result was like: (Dendrogram plot)



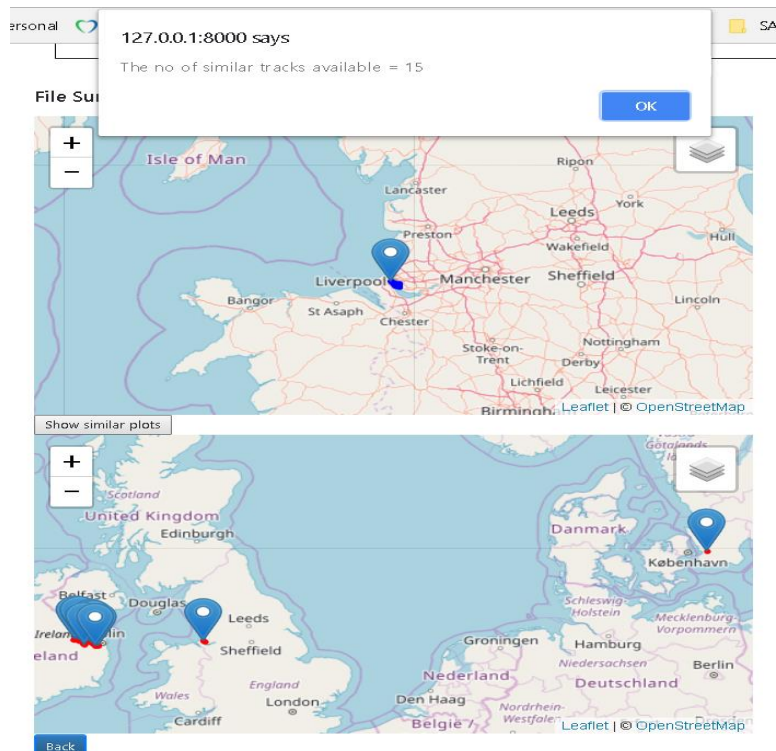
It is evident from the 2 algorithms execution that when we are grouping in 4 sets there will be 2 small groups containing 3 files.

We are showing similar paths with respect to a chosen route as per k-means output. In the star plot the file 'activity_2595492741.gpx' represents the 1st green star from the left-hand side. So, if we choose this file in the dropdown, we will get 2 other similar paths in the similar plot (in total 3 routes).

The actual result in the webpage is like:



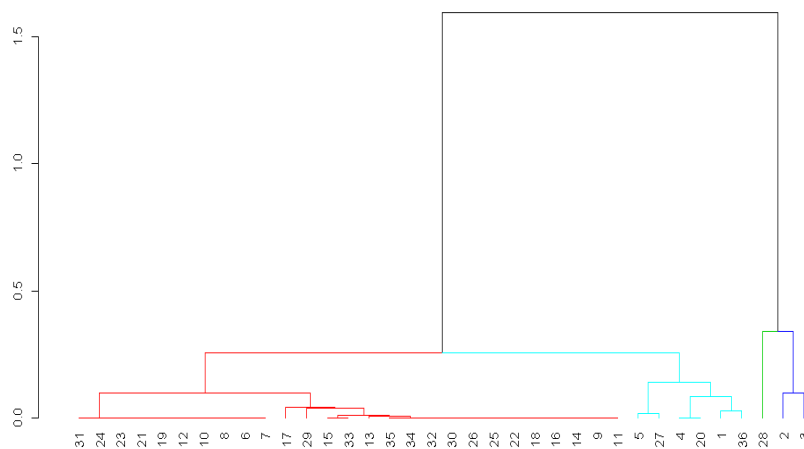
Similarly, if we choose any file that falls within any 15 files group, we will get a similar plot with 15 routes in it. Say we select the file 'activity_1653497591.gpx' (the 1st file marked in red [star plot]). The result would be like:



Now say we want to only consider angle and discard all other field values while clustering dataset into groups. We did little changes in the dendrogram section like:

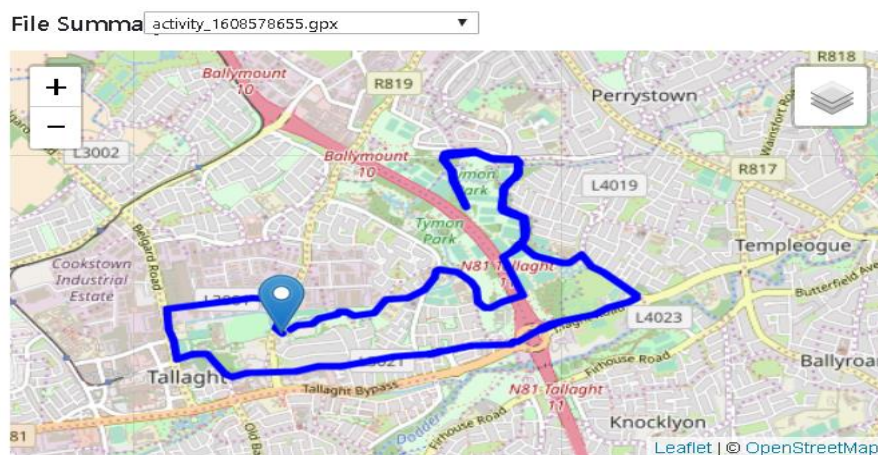
```
d<- dist(dataSet2Analysis[,8],"euclidean")
h<- hclust(d,"average")
d1 <- as.dendrogram(h)
d2=color_branches(d1,k=4,col=c(2,5,3,4)) # auto-coloring 4 clusters of branches.
plot(d2)
dev.copy(jpeg,filename="C:\\subhajit\\projectX\\smartblog\\smartblogproject\\Plots\\DendoPlot.jpg")
dev.off()
```

Here the 8th field represents the angle in dataset2Analysis. The output we got:

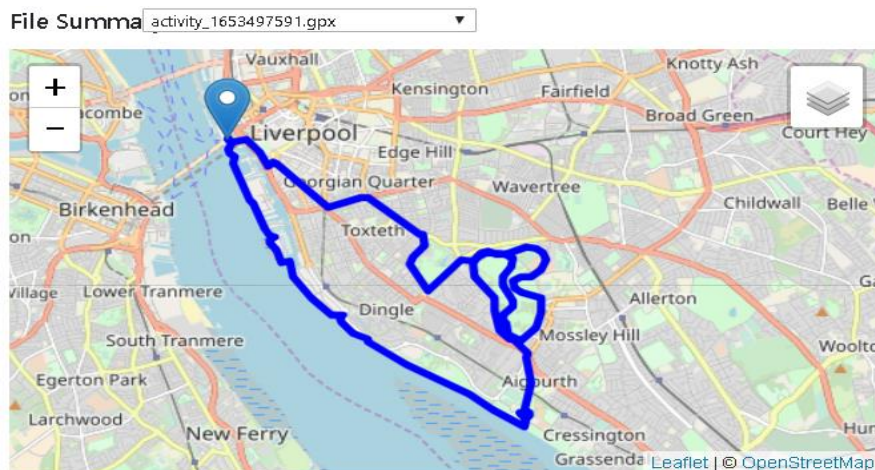


Here we can visualize that 2nd & 3rd files shapes should be similar. Let's see that in map plot.

2nd files(activity_1608578655.gpx) shape:



3rd files(activity_1653497591.gpx) shape:



We can witness that the shapes are identical. So, on an individual field basis, we can cluster, files into relevant groups as well.

CHAPTER 5: Conclusions and Future Work.

Section 5:1 Summary of Thesis:

Right from deciding the thesis project objectives to the completing of the project work, each of the steps is documented and explained in this thesis paper. The motivation of doing this project was to build something meaningful, which would be relevant for general citizens' use and to help them find suitable sports activity tracks as per their choice. It will encourage them to achieve better health standards.

We divided our documentation in 4 prime chapters. The first chapter explains our goals of this thesis, how we analysed the problem statements and came up with solutions. It gives a brief idea about how we realized which field of knowledge were essential to build up the solution, and how different fields of expertise were adhered together to secure the ultimate goals.

In chapter 2, a brief research work was explained. The study of different published academic papers and the conventions that can be helpful to achieve project objectives, were considered. A list of software or applications available on the internet were scrutinized and gathered inspiration from them to design an application, which would be even more relevant for user benefits with added functionalities. We developed a web-based application which groups together similar kind of trails, from the defined characteristics. We merged together machine learning concepts, web development expertise and database knowledge to build the product.

One of the most important chapters is chapter 3. Elaboratively, the solution design of the product was described with diagrams. From the data acquisition to final output rendering in webpages, each step of the process flow were explained with reasons. In this chapter, it was explained what concepts or methodologies were used, why they were used and how they were used.

Chapter 4 summarizes the outcome of the project work. The results obtained from the designed solution is documented and analysed here. A different set of GPX files were referred at different points and results were compared. A big dataset with 36 GPX files were thoroughly processed, examined and after executing data science related algorithms the outputs were analysed. Comparisons were made between result sets obtained from different machine learning algorithms execution and the relevance of each resultant to the user is explained in this module.

Section 5:2 Overall Evaluation

The objectives discussed in chapter 1.2 and chapter 4 are met successfully in this thesis. The GPX data is analysed in all possible ways and derived fields as much as possible, which led us to figure out the most successful similar clustering of tracks. The solution pattern and architecture were discussed in the previous chapters and their validity is cross verified with the outcome achieved.

As we were focused to build a web application, there should be minimum manual effort involved in the application flow. But after loading PostgreSQL table with GPX file content, we still require executing the discussed R script file manually. Attempts were made to automate this step but with no success. The actions are detailed below:

- Tried to install the rpy2 package: As found in various tutorials and forums, we tried to install rpy2 package within the local system, but it did not work. ([readthedocs](#), n.d.))
- Tried with subprocess command: As per the site ([r-bloggers](#), n.d.)) we try to execute the R script but it did not work.

Section 5:3 Future Work

It would be an excellent opportunity to propose some future work utilizing the platform built in this thesis work. The web application built, is providing functionalities locally. By using cloud storage like AWS S3 and hosting the application in Heroku like platform, we will be able to access the application publicly. Then the users will be able to upload their GPX files globally and through machine learning concepts utilization, will find tracks suitable for

defined characteristics. We mainly used unsupervised machine learning in this project. But if we blend supervised machine with the existing implementation we can achieve below mentioned 2 objectives.

- 1) **Fitness Program:** In this modern and fast paced age, people are realizing how important it is to be physically active and fit. Even doctors give huge importance to fitness programs nowadays. As they say, prevention is better than cure. Like celebrities, common people also are getting enthusiastic about setting up fitness goals. If we can access fitness record of individuals and their interest in sports activities (like cycling, hiking, running, kayaking etc), we can suggest routes available near their stay or choice of location to achieve desired physical fitness.
- 2) **Building suitable tracks for Sports activity:** Governments and big organizations are continuously making effort to encourage their citizens or employees to be physically active and fit as it is statistically proven that it helps to create a healthy society and enhance productivity. Governments built tracks only meant for certain sports activities like cycling, hiking etc. But many of the construction does not get utilized as they were anticipated at the start. So, fetching the spatial data of any location, activity of the people and government or companies future construction plan, we will be able to guide them to build cost effective trails and eventually help them to build smart roads or cities.

APPENDICES

Access to the codebase:

Dump the package list

Upload in github

Filepath(image) declaration

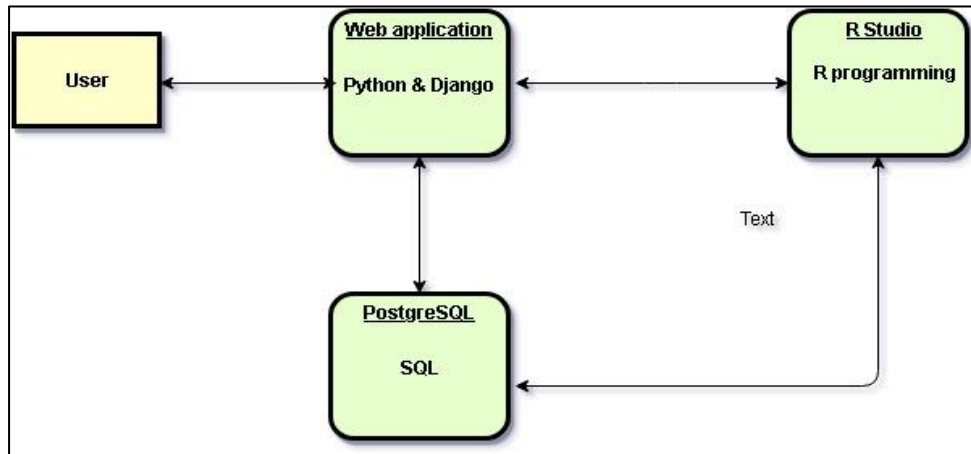
Create instruction

In this section we will try to brief some of the concepts that we used while development, along with solution architecture that we build. We will also point out some the most significant code snippets.

Application architecture:

As our one of prime objective 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)

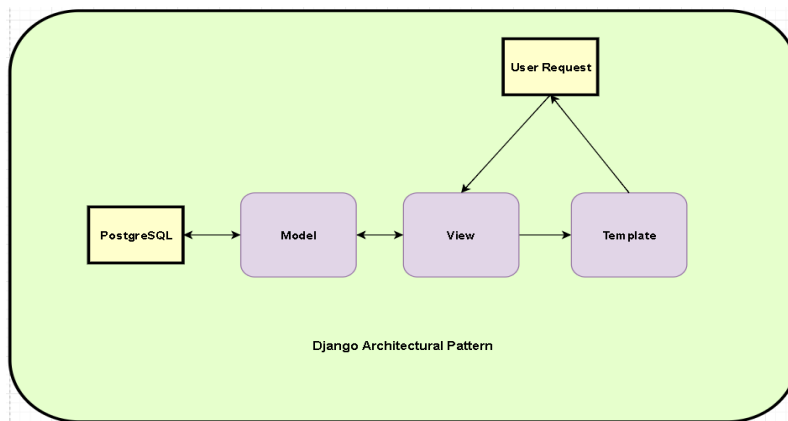
PostgreSQL, and 3) R programming. Having said that, some other technologies or conventions like REST-API, Ajax, Javascript, JQuery library, Bootstrap library etc played a significant role in making our project a success. They will also be discussed as per the need.



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).

As we used a python web framework named Django, here is a brief description of its architectural pattern.

Django Architectural Pattern:



Django accompanies the MVC pattern very closely, however it has its own conventions for certain implementations. As the “Control (C)” is handled by the framework itself and most of the excitement in Django is defined in models, templates and views, Django is often termed as an *MTV framework*. In the MTV development pattern:

- **M - “Model,”** the data access layer. This layer holds everything about the data: data access, data validation, defining state of it, and the relationships between the data.
- **T - “Template,”** the visualization layer. This layer consists of presentation-related decisions: how something should be displayed on a Web page or other form of document.
- **V - “View,”** the business logic layer. This layer consists of the logic that accesses the model and renders the derived response to the appropriate template(s). It acts as the bridge between models and templates.

Django’s view is more like the controller in MVC, and MVC’s view is a Template in Django. ((djangobook, n.d.))

Walk through of the Web Application & code snippets:

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. Before detailing each section, we will try make out how Django framework works.

The admin section is good enough to create a user and to provide the basic authentication and authorization for different users

Django administration

Site administration

AUTHENTICATION AND AUTHORIZATION		
Groups	+ Add	Change
Users	+ Add	Change
RESTAPP		
Employeeess	+ Add	Change
Gpx jsons	+ Add	Change
Gpxcontent tables	+ Add	Change
Summary tables	+ Add	Change
SMARTBLOGAPP		
Posts	+ Add	Change

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.

Django administration WELCOME SUBHAJIN VIEW SITE / CHANGE PASSWORD / LOG OUT

Home : Smartblogapp > Posts > Mountaineering

Change post HISTORY VIEW ON SITE

User: subhajit [+](#) [-](#)

Title: Mountaineering

Image: Currently: None/Mountaineering_AJFDvId.jpg [Clear](#)
Change: [Choose File](#) No file chosen

Height field: 183

Width field: 275

Content:

Mountaineering is the sport of mountain climbing. While some scholars identify mountaineering-related activities as climbing (rock and ice) and trekking up mountains [2] others are also adding backpacking, hiking, skiing, via ferrata and wilderness activities [3] and still others state that mountaineering activities also include indoor climbing, sport climbing and bouldering [4]. However, to most of the scholars, the term mountaineering is understood as climbing (which now refers to adventure climbing or sports climbing) and trekking (hill walking in 'exotic' places) [5][6]. Hiking in the mountains can also be a simple form of mountaineering when it involves scrambling, or short stretches of the more basic grades of rock climbing, as well as crossing glaciers.

Latitude: 0.0

Longitude: 0.0

[Delete](#) [Save and add another](#) [Save and continue editing](#) [SAVE](#)

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:

running



1 week ago

Running is the way in which people or animals travel quickly on their feet. It is a something of travelling on lan...

[View](#)

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:



Details Aug. 7, 2018, 12:37 p.m.

Author: subhajit

1	Edit Post	Delete Post
Running		
Running is the way in which people or animals travel quickly on their feet. It is a something of travelling on land. It is different to walking in that both feet are regularly off the ground at the same time.[1] Different terms are used to refer to running according to the speed: jogging is slow, and sprinting is running fast. Running is a popular form of exercise. It is also one of the oldest forms of sport. The exercise is known to be good for health; it helps breathing and heartbeat, and burns any spare calories. Running keeps a person fit and active. It also relieves stress. Running makes a person thirsty, so it is important to drink water when running.		
<input type="text" value="C:\gpxFiles\"/> <input type="button" value="Upload GPX Files"/>		
<input type="button" value="Load Summary API"/>		

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.

when running.

C:\gpxfiles\

Upload GPX Files

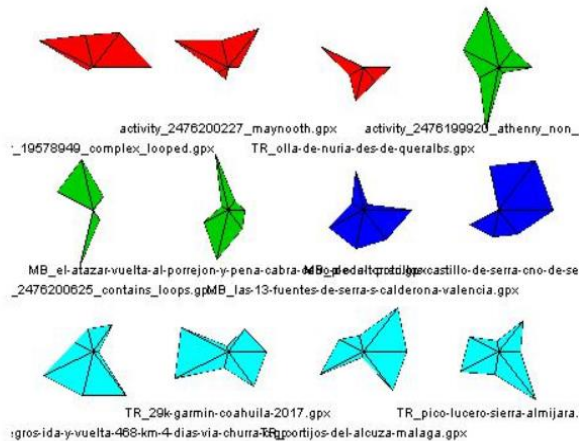
Load Summary API

Files Summary

Elbow Curve

Clusters

Summary Table



Files Summary

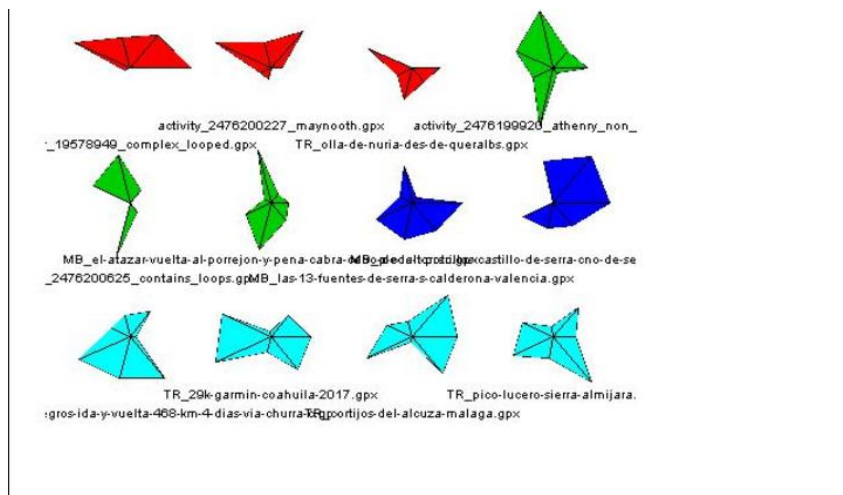
Elbow Curve

Clusters

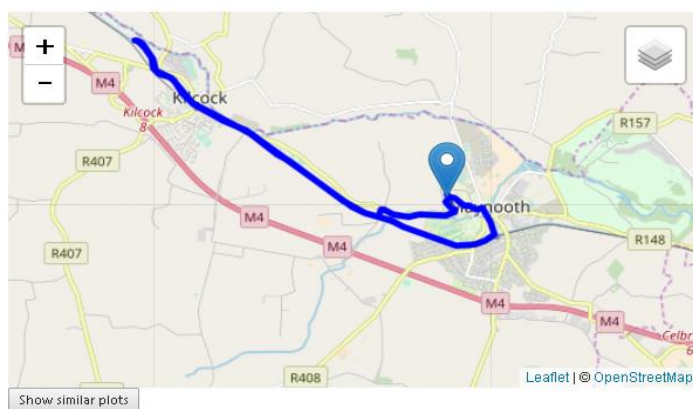
Summary Table

File Name	Altitude	Time	Speed	TimeDiff	Delta Elevation	Total Distance	Angle
activity_19578949_complex_looped.gpx	7.2	-4140	3.5597	6	1.7999992	13.9970697406647	103.3
activity_2476199920_athenry_non_loop.gpx	3.4	-2016	4.9711	4	3	9.99606886798002	103.8
activity_2476200227_maynooth.gpx	13.2	-4785	3.5579	6	1.8	16.4300179978495	103.3
activity_2476200625_contains_loops.gpx	-1.8	-1666	4.831	4	1.600002	8.017681288932	103.9
MB_el-atazar-vuelta-al-porrejón-y-pena-cabra-cerro-de-la-torre.gpx	1593.3	-14859	4.85123	17	30.287	50.8970640758572	111.2
MB_las-13-fuentes-de-serra-s-calderona-valencia.gpx	640.6	-12705	4.064	19	34.185	41.1919688265815	112.3
MB_p-coeli-potrillos-castillo-de-serra-cno-de-serra-agatova-co.gpx	652.2	-8899	4.3893	16	31.169	28.8113634347491	111.8
MB_valencia-ojos-negros-ida-y-vuelta-468-km-4-dias-via-churra-c.gpx	1196.31	-216740	2.5346	300	71.126	455.666740052818	113.1
TR_29k-garmin-coahuila-2017.gpx	3424.6	-20403	1.4801	50	74.706	27.4012593138775	112.52
TR_cortijos-del-alcuza-malaga.gpx	560.28	-8084	2.3019	33	48.582	14.0349765544569	113
TR_olla-de-nuria-des-de-queralbs.gpx	2843.6	-1136879	1.029192	46	1423.386	388.188994845428	108.9
TR_pico-lucero-sierra-almijara.gpx	1763.4	-25606	1.7473	36	48.373	28.5172457509234	115.1

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.

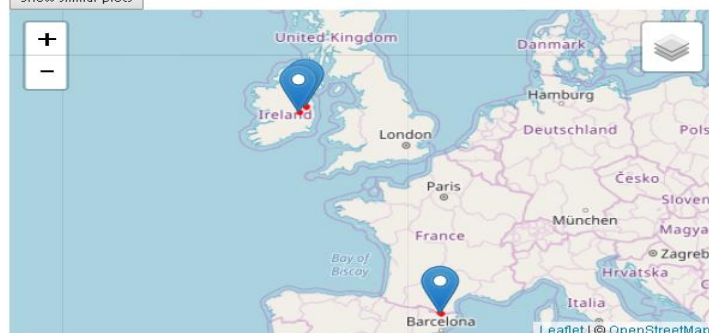
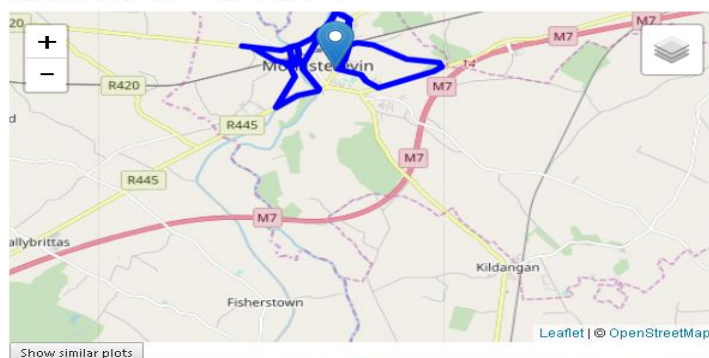


File Summa activity_2476200227_maynooth.gpx



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.

File Summa activity_19578949_complex_looped.gpx



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.

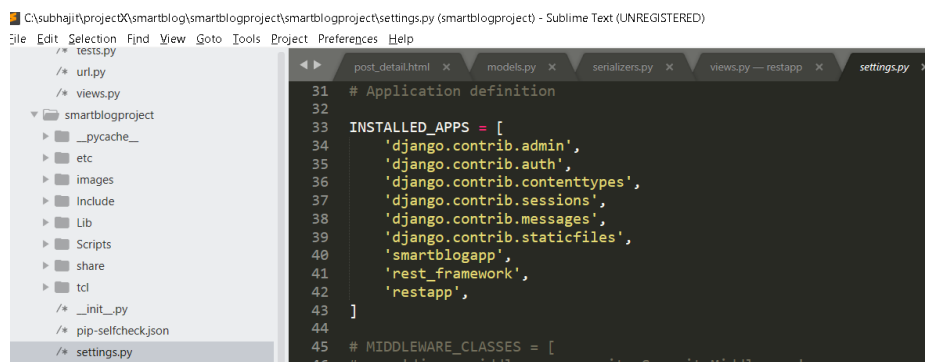
User will be able to store the relevant data and execute machine learning algorithm as per different activities. The above example was regarding 'running' activity and the relevant files were process.

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.

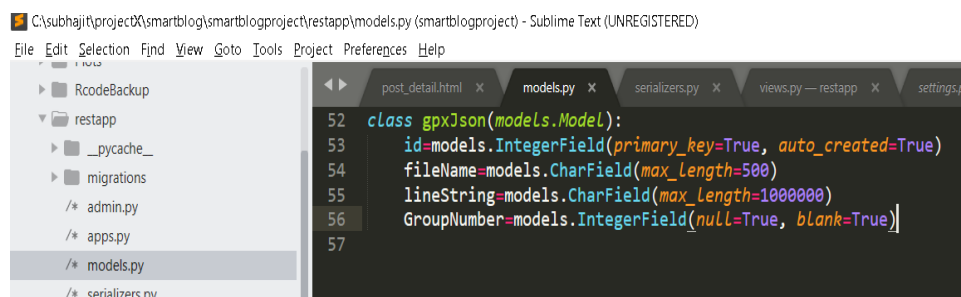


```
C:\subhajit\project\smartblog\smartblogproject\smartblogproject\settings.py (smartblogproject) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

/* tests.py
/* url.py
/* views.py
smartblogproject
├── __pycache__
├── etc
├── images
├── Include
├── Lib
├── Scripts
├── share
├── tcl
├── __init__.py
├── pip-selfcheck.json
└── settings.py

31 # Application definition
32
33 INSTALLED_APPS = [
34     'django.contrib.admin',
35     'django.contrib.auth',
36     'django.contrib.contenttypes',
37     'django.contrib.sessions',
38     'django.contrib.messages',
39     'django.contrib.staticfiles',
40     'smartblogapp',
41     'rest_framework',
42     'restapp',
43 ]
44
45 # MIDDLEWARE_CLASSES = [
46     'django.middleware.security.SecurityMiddleware',
```

- 2) 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.



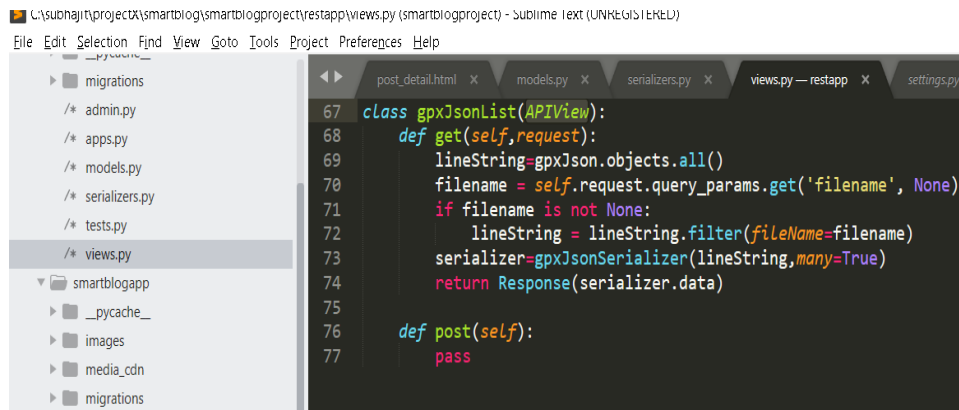
```
C:\subhajit\project\smartblog\smartblogproject\restapp\models.py (smartblogproject) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

RcodeBackup
restapp
├── __pycache__
├── migrations
├── admin.py
├── apps.py
├── models.py
└── serializers.py

52 class gpxjson(models.Model):
53     id=models.IntegerField(primary_key=True, auto_created=True)
54     fileName=models.CharField(max_length=500)
55     lineString=models.CharField(max_length=1000000)
56     GroupNumber=models.IntegerField(null=True, blank=True)
57
```

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.

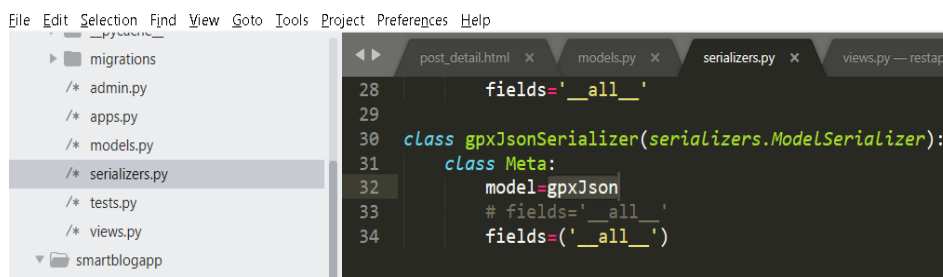
- 3) Create a view class for each model class created. The response of the view class should be a APIView (type) which derived from 'rest_framework.views'.



```
67 class gpxJsonList(APIView):
68     def get(self, request):
69         lineString=gpxJson.objects.all()
70         filename = self.request.query_params.get('filename', None)
71         if filename is not None:
72             lineString = lineString.filter(fileName=filename)
73         serializer=gpxJsonSerializer(lineString,many=True)
74         return Response(serializer.data)
75
76     def post(self):
77         pass
```

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.

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



```
28     fields='__all__'
29
30 class gpxJsonSerializer(serializers.ModelSerializer):
31     class Meta:
32         model=gpxJson
33         # fields='__all__'
34         fields=('__all__')
```

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.

- 5) 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:

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

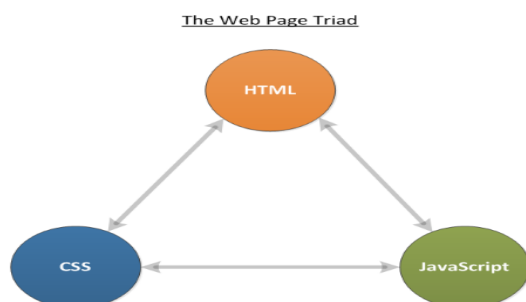
The whole purpose of creating REST-APIs were:

- 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.

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 constructs the page**, including the text, graphics, etc. CSS is used to design the look and feel 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 through the access of DOM objects.



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.

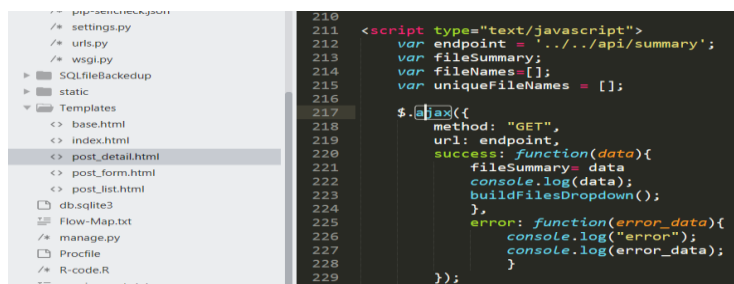
Jquery:

jQuery is a library of Javascript functions and utilities that add visual effects and make advanced features simple to implement in just a few lines of code. It is supported across all major browsers, and open source.

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.

jQuery is used for the below mentioned advantages:

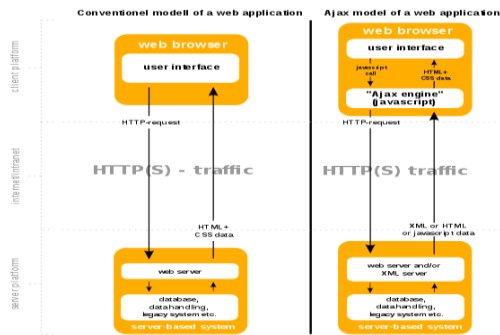
- 1) **Defines Easy events:** Nearly all software works on an event model – click on something, and a click event is triggered. Drag your finger across a tablet, and a drag event is triggered. Applications “listen” for these events and do something – jQuery lets you do this in a browser.
- 2) **Perform AJAX call Simple:** 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 to access web servers from the client side. 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. ((wikipedia, n.d.))



A sample AJAX code is found below: (w3schools, n.d.)

```
<script>
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("demo").innerHTML = this.responseText;
    }
  };
  xhttp.open("GET", "ajax_info.txt", true);
  xhttp.send();
}
```

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.

Summarising academic papers

Sports as it is known occupies one of the largest spaces in the list of activities that humans like and actually love to spend their time in. As evident as it is with the extreme propagation of digital devices and mobile phones; all of which include global positioning systems sensors widely these days have revolutionized the level of information available for us to store for one of our most liked activities; again sports. GPS data has provided geographic knowledge that have been important in design of novel technology. Some works use GPS data for several goals in health sciences (Nancy F Butte, 2012) and agronomic sciences (E Costa, 2007).

But Since the increase in popularity of GPS devices, they are widely used for tracking sport activities. Recognizing the transportation modes of people is an open research problem with applications in fields such as pervasive computing and mobile applications. The mobile phones of previous generation included different kinds of sensors that were capable of registering the real-time information of users such as position, co-ordinates, speed, etc. The most amazing one among them were the GPS receiver that made it possible for users to spatially find the device and actually position-track it on a regular basis. Needless to say; this data was of significant importance as it lays the foundation for producing innovative applications to the device user. For example, if a mobile or the related digital device is capable of registering the beginning of a user who has started their running

session, it could automatically set a specific configuration programmed to enhance the user interaction with that context which may include features like noise level, phone interface etc . (DSIC, 2016).

The use of GPS tracks in machine learning has been employed for different purposes. Identifying the movements related to modes of transportation of people have been known to have significant applications in pervasive computing (Armanda Rodrigues, 2014). Popular apps such as Runtastic or Endomondo that have been there for a while now have allowed users to keep a log of the tracks of their own activities after which the user has an option to share their information on social networks. For instance; Wikiloc1 is one such network. MapMyTracks2 and everytrail3 are other such related examples.

While on networks like Wikiloc1; there are more than a million users who have shared about three million GPS tracks of more than seventy different kinds of sports. These sports are the ones with rapid dynamic movements and can be Sailing, Hiking, Cycling and so on. These GPS tracks are located all around the world. On networks like MapMyTracks2 and everytrail3 every time a user uploads a new track, they must introduce the kind of sports activity for the GPS track manually.

REFERENCES/BIBLIOGRAPHY

- .r-bloggers, n.d. [Online]
Available at: <https://www.r-bloggers.com/integrating-python-and-r-part-ii-executing-r-from-python-and-vice-versa/>
[Accessed 16 08 2018].
- Armanda Rodrigues, C. D. a. J. E. C., 2014. In: *In Connecting a Digital Europe Through Location and Place*. s.l.:s.n.
- djangobook, n.d. [Online]
Available at: <https://djangobook.com/model-view-controller-design-pattern/>
[Accessed 19 08 2018].
- djangoproject.com, n.d. *Django Documentation*. [Online]
Available at: <https://docs.djangoproject.com/en/2.1/>
[Accessed 19 08 2018].
- DSIC, U. P. d. V., 2016. In: *Identifying the Sport Activity of GPS Tracks*. s.l.:s.n.
- E Costa, A. L. A. C. a. A. F., 2007. In: *Evaluation Methods for Machine Learning II*. s.l.:s.n.
- Nancy F Butte, U. E. a. K. R. W., 2012. In: *Medicine and science in sports and exercise*. s.l.:s.n.
- readthedocs, n.d. [Online]
Available at: https://rpy2.readthedocs.io/en/version_2.8.x/introduction.html
[Accessed 16 08 2018].
- stanford.edu, n.d. *An Introduction to statistical learning*. [Online]
Available at: https://web.stanford.edu/~hastie/ElemStatLearn//printings/ESLII_print10.pdf
[Accessed 02 08 2018].

stanford.edu, n.d. *The Elements of Statistical Learning*. [Online]

Available at: https://web.stanford.edu/~hastie/ElemStatLearn//printings/ESLII_print10.pdf
[Accessed 01 08 2018].

w3schools, n.d. [Online]

Available at: https://www.w3schools.com/xml/ajax_xmlhttprequest_response.asp
[Accessed 19 08 2018].

wikipedia, n.d. [Online]

Available at: https://en.wikipedia.org/wiki/GPS_Exchange_Format
[Accessed 16 08 2018].

wikipedia, n.d. [https://en.wikipedia.org/wiki/Ajax_\(programming\)](https://en.wikipedia.org/wiki/Ajax_(programming)). [Online].