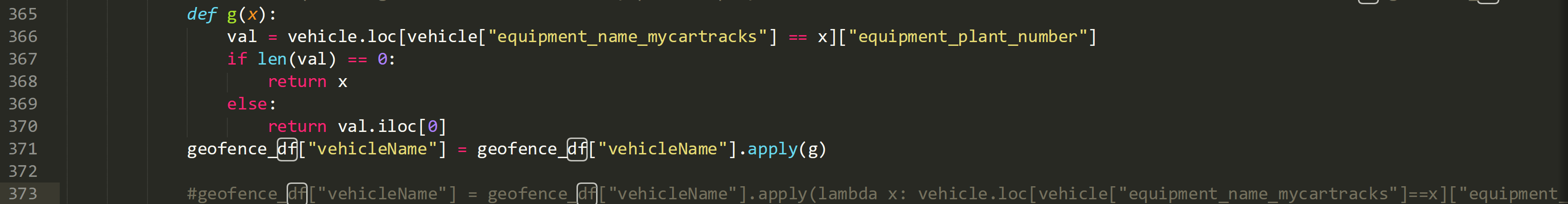
# Développement à finaliser

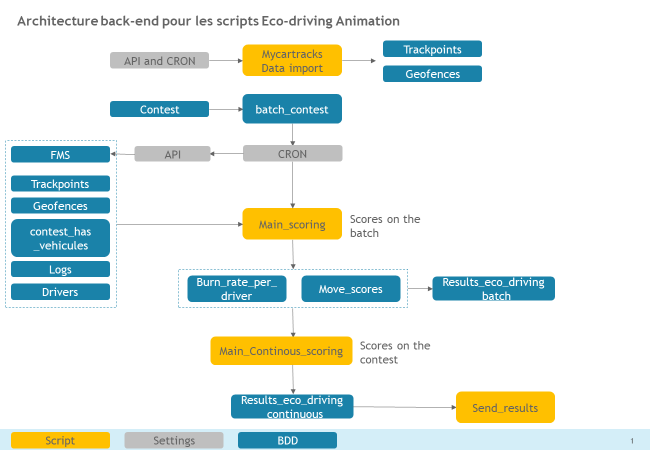
* Etats des lieux :
  + Absence de concours sur Sadiola (en attende de la signature du contrat)
    - Le script est toujours fonctionnel sur le précédent concours
  + Préparation du lancement d’un nouveau concours sur Bonikro
    - Pourquoi les géofences de Bonikro ne remontent des données uniquement pour le véhicule « T51-SFTP » ? -> sur le script MyCarTracks Data Import, correction de la ligne 373 par la fonction g



* + Préparation du lancement d’un premier concours sur CBG
    - Paramétrage des géofences de maintien de la vitesse @ Eloi
    - Paramétrage de l’interface de maintenance @Eloi
    - Résolution du ticket JIRA
* **Must have** :
  + Robustifier le script si absence de données
    - Cas de Bonikro : absence de données de géofences provoquant l’arrêt du script car le score « speed maintenance » ne donne rien (try – except ?)
  + Tester le script pour la mine de CBG :
    - Industrialiser la mise en place du pivot pour intégrer les identifiants WENCO dans la table « appid\_operations» @Hamza et @Loïc
    - Mise en place du score multi-geofence pour le score «speed\_maintenance » une fois les geofences paramétrées
  + « Réparer » le CRON pour qu’il tourne automatiquement (en attendant le lancer à la main tous les lundis)
  + Adapter la version multi-pit du climb and descent à Bonikro
    - A industrialiser à partir du notebook Jupyter
* **Nice to have** :
  + Le score “speed\_maintenance” fonctionne bien (vérifier la cohérence sur Jupyter)
  + Le score « consumption\_per\_driver» ne fonctionne plus (retester sur Bonikro concours passé)
  + Le score « move\_score\_all\_stop\_points» fonctionne bien (vérifier la cohérence des courbes sur Jupyter)
  + Agréger les scores « climb and descent » et « speed maintenace» pour obtenir un unique score de maintien de la vitesse pour les véhicules jouant le rôle à la fois de « Dumper » et de « Haulage »
  + Commencer les notices des scripts et tester que les CRONs fonctionnent bien :
    - Main\_Continuous\_scoring
    - Send\_results
  + Finaliser l'intégration des sanity checks
  + Stocker les images (courbes d’accélération, de décélération) dans le serveur pour vérification

# Scoring script notice

Le fonctionnement général du script est illustré dans l’image ci-dessous :

**

1. **Scores used for each mine**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Scores | Consumption per driver | Speed maintenance | Move score for specific stop points | Move score for all stop points | Climb and descent |
| Bonikro:  type= ”Haulage” | X | X |  | X |  |
| Bonikro:  Type= “Dumping” | X |  |  | X | X |
| Sadiola:  Type=”Haulage” |  |  |  | X | X |
| Bissa  Type = “Haulage” |  | X |  | X | X? |
| CBG  Type = “Haulage” |  | X |  | X | X? |
|  |  |  |  |  |  |

1. **Scripts:**

The scripts can be found in the following file:

1. Data upload:

***‘…\Dropbox\Jupyter\ScriptsOVH\eco driving\scoring\0.MycCarTrackDataImport\MyCarTracks\_data\_import\_v11\_Bonikro.py’***

***‘…\Dropbox\Jupyter\ScriptsOVH\eco driving\scoring\\0.MycCarTrackDataImport\MyCarTracks\_data\_import\_v11\_Sadiola.py’***

***‘…\Dropbox\Jupyter\ScriptsOVH\eco driving\scoring\\0.MycCarTrackDataImport\MyCarTracks\_data\_import\_v11\_Bissa.py’***

***‘…\Dropbox\Jupyter\ScriptsOVH\eco driving\scoring\\0.MycCarTrackDataImport\MyCarTracks\_data\_import\_v11\_CBG.py’***

1. Scoring script:

***‘…\Dropbox\Jupyter\ScriptsOVH\eco driving\scoring\Main\_scoring\_v10\_Bonikro.py’***

***‘…\Dropbox\Jupyter\ScriptsOVH\eco driving\scoring\Main\_scoring\_v10\_Sadiola.py’***

1. CRON:

***‘…\Dropbox\Jupyter\ScriptsOVH\eco driving\scoring\CRON\_MyCarTracks\_data\_import.py’***

***‘…\Dropbox\Jupyter\ScriptsOVH\eco driving\scoring\CRON\_batch\_contest\_v3\_Bonikro.py’***

***‘…\Dropbox\Jupyter\ScriptsOVH\eco driving\scoring\CRON\_batch\_contest\_v3\_Sadiola.py’***

1. **Parameters:**

The main parameters we use are:

* The date
* The API and database Ids and passwords
* The contest specification parameters:
  + The type of vehicle to consider (i.e. either Haulage or DT)
  + Contest Id
  + Batch number
  + Batch duration

1. **Global architecture:**

The algorithm is three-fold:

1. Data fetching

* We fetch the beginning and ending dates and put them to the right format (%Y%  
  m%d:%H/%M/%S).
* We fetch the drivers’ info data
* We focus on the contest environment. i.e. we focus on the vehicles to consider for this specific contest.
* We fetch the FMS data
* We fetch the geofence and track-points data

1. Transformation

The inputs we take in are the following: drivers data, drivers names, fms data, track-points data, geofence data, vehicles to consider

* **Function 1: Consumption per driver**

**Objective:** we compute the “burn rate” score for each driver

We get the hours driven by each driver between two refills.

* We create the refills per vehicle
* We create the consumption per driver
  + We remove the drivers who have not driven enough hoursd

To have a better grasp on it, you could run it on Jupyter using the below scripts:

**‘…\Dropbox\Jupyter\Notebooks\Eco-driving - Henri\ 4.Consumption\_per\_driver\_VPS\_v3.ipynb’**

* **Function 2: Speed maintenance**

**Objective**: we compute the speed maintenance in geofences where the vehicle is supposed to maintain a steady speed

* + We select the vehicles to consider
  + We select the ‘Cruise Speed’ geofence
  + We map the drivers’ names to the cruise speed data (i.e. to all geofence data reduced to the universe we want to consider)
  + We format the EntryTime and ExitTime dates

Then we get the Cruise speed per driver: i.e. we get the mean and std of speed over the EntryTime-ExitTime period.

We get two scores: speed score, which is the trimmed std of speed, and speed stability (which is a vector of mean(speed) and std(speed))

To have a better grasp on it, you could run it on Jupyter using the below scripts:

**‘…\Dropbox\Jupyter\Notebooks\Eco-driving - Henri\5.Move\_scores\_and\_speed\_maintenance\_VPS\_v3.ipynb’**

* **Function 2 bis : Speed maintenance avec multi-géofence**

**Lien Notebook**

<http://localhost:8888/notebooks/Dropbox/Jupyter/Notebooks/Eco-driving%20-%20Hamza/Multi-Goefence/Main_scoring_v10_CBG_multigeo_vfinale.ipynb>

Pour obtenir le multi\_geofence, le code modifié est principalement celui du « speed\_maintenance » (fonction déjà existante et appliquée pour le haulage)

On passe d’un calcul unique, à une boucle sur les geofence, suivie d’un calcul d’agrégation des scores pour en obtenir un unique. La pondération se fait sur le temps passé dans les geofences.

On commence donc par une boucle sur les geofences : (début du code speed\_maintenance)

*NB : Les geofences considérées sont dans les paramètres, à la clé ‘multi-geo’*

* Une boucle calcule le score pour chaque driver sur la geofence
* Output par driver : driver\_name, score, nombre de mouvements, vitesse moyenne, temps passé dans la geofence, id\_batch, id\_contest

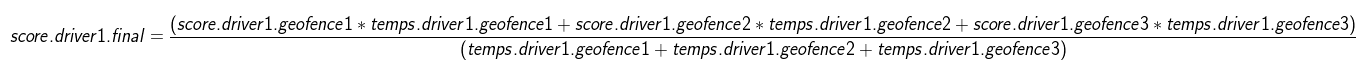
Cette boucle est réalisée pour toutes les geofences présentes et le résultat final est un tableau ayant pour colonnes : driver\_name, score, nombre de mouvements, vitesse moyenne, temps passé dans la geofence.

Les lignes de ce tableau sont la concaténation des lignes de toutes les géofences, c’est-à-dire qu’on rassemble sous le même tableau tous les scores des drivers pour toutes les géofences.

Donc dans ce tableau, certains drivers apparaissent sur plusieurs lignes (car ils sont passés par plusieurs geofences).

Donc il faut merger les lignes :

* On commence par la moyenne pondérée des scores en fonction du temps passé dans les géofences (formule ci-dessous) :

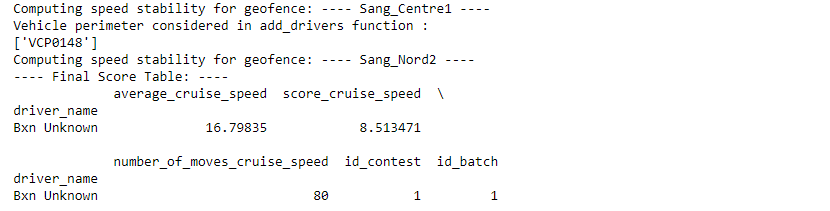


*NB : on a fait la conversion du temps qui était au format hh :mm :ss en temps en secondes pour le calcul de la moyenne*

* Ensuite calcul de la vitesse moyenne pondérée par le temps passé dans les geofences (même formule que ci-dessus)
* Enfin calcul du nombre total de mouvement du driver (somme simple)

On a enfin un tableau de résultat final ci-dessous, chaque ligne est le score pondéré par driver par géofence.

Tableau de résultat :



* **Function 3: Move score for specific stop points**

**Objective**: We compute the acceleration and braking score at specific stop points

We reduce our scope to the vehicles to consider and the relevant trackpoints

Step 1: Finding all stop points

***A stop point is defined as***: all point which is close to the coordinates given where the vehicule is spotted with zero speed.

First, we set the longitude and latitude parameters to focus on the vehicles to consider (i.e. we only keep the track points data which concerns the contesting vehicles)

We drop the duplicates.

We clean our data by removing the spot points which can be accounted for by a loading or an unloading.

We determine the geofence where the vehicle entered after its stop point and at what time.

We delete the unnecessary stop points (if more than one for the path) and qualify each stop.

Acceleration for the first stop points

Braking for the last ones

Step 2: Computing the scores

Acceleration Braking:

We use our stop points

We plot the movement curves (the speed curves) and a reference curve which is the average speed curve on a vehicle over the different drivers. We focus on the 60 seconds around the measurement point.

The score is: trimmed mean of the aggregate disparity over all considered points. Where the disparity is defined as the “distance” or difference between the actual speed curve and the refence speed curve.

To have a better grasp on it, you could run it on Jupyter using the below scripts:

**‘…\Dropbox\Jupyter\Notebooks\Eco-driving - Henri\5.Move\_scores\_and\_speed\_maintenance\_VPS\_v3.ipynb’**

* **Function 4: Move score for all stop points**

**Objective**: We compute the acceleration and braking score at specific stop points

Input parameters:

* We set the minimum time a vehicle has to be stopped in seconds to be considered a clear stop (we set it to 60 seconds)
* We set the maximum speed a vehicle can reach to be considered stopped (we set it at 2Km/h)
* We set the speed a vehicle needs to reach after a stop to consider it is not stopped anymore.

We use these criteria to get the stop points.

The scores are then obtained in the same way as they are computed in Function 3/ Step 2

To have a better grasp on it, you could run it on Jupyter using the below scripts:

**‘…\Dropbox\Jupyter\Notebooks\Eco-driving - Henri\** **Move\_scores\_all\_stop\_points\_VPS.v2.ipynb’**

* **Function 5: Climb and descent**

**Objective**: we compute the speed maintenance in detected climb and descent

Input parameters:

* We set the minimum time a vehicle has to be stopped in seconds to be considered a clear stop (we set it to 60 seconds)
* We set the maximum speed a vehicle can reach to be considered stopped (we set it at 2Km/h)
* We set the speed a vehicle needs to reach after a stop to consider it is not stopped anymore.

We use these criteria to get the stop points.

The scores are obtained the same way they are computed in Function 3. Except for a small detail: the scores are weighted.

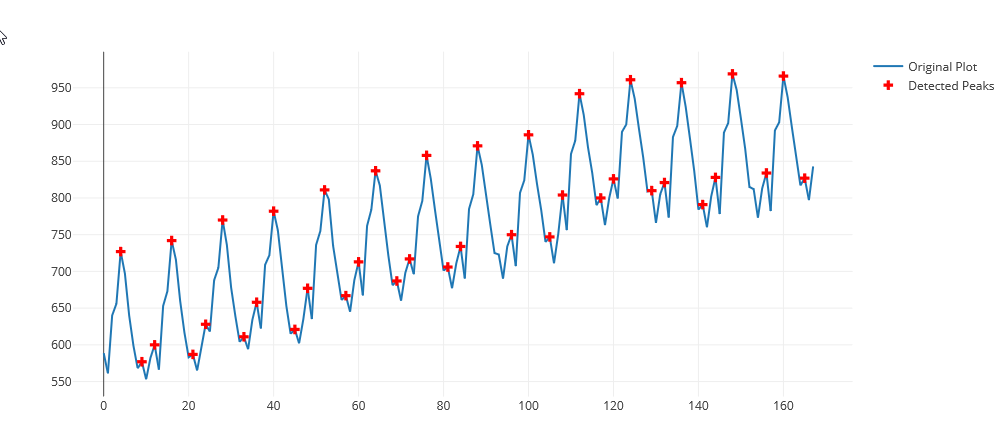
Score descent = (score descent- min(score descent))/(max(score descent)-min(score descent))

Score climb= (score climb-min(score climb))/(max(score climb)-min(score climb))

Score cruise speed=(#descent \* score descent + #climb\*socre climb)/(#descent + #climb)

For the speed maintenance DT we use the same logic as that of Haulage type vehicles, except that the DT type vehicles are moving in and out of a pit, so we need to factor this parameter in to compute the speed maintenance score. We do the following:

We determine the peaks and the bases these vehicles reach over time: i.e. we take their altitudes over time and spot these points. Here is an example to make it clearer:



Once we get these points, we perform our analysis on a 20s before the middle and 20s after the middle for the climbs and 40s before the middle and 40s after the middle for the descents. A “middle” is the middle point between a peak and a base.

Please refer to the following websites should you have any further questions about the peakutils package:

* *https://plot.ly/python/peak-finding/*
* *https://pythonhosted.org/PeakUtils/reference.html : features of peakutiles.indexes()*

To have a better grasp on it, you could run it on Jupyter using the below scripts:

**‘…\Dropbox\Jupyter\Notebooks\Eco-driving - Henri\** **Climb\_and\_descent\_VPS\_v2.ipynb’**

* **Function 5 bis : Climb and descent multi-pit**

Multi-pit + moyenne pondérée

**Lien Notebook**

<http://localhost:8888/notebooks/Dropbox/Jupyter/Notebooks/Eco-driving%20-%20Hamza/Multi-pit/Main_scoring_v10_Bonikro_multipit_vFinale.ipynb>

C:\Users\loic\Dropbox\Jupyter\Notebooks\Eco-driving – Henri\Climb\_and\_descent\_VPS\_v2\_MODIF\_SDC.ipynb

Pour obtenir le multi\_pit, le code modifié est principalement celui du « climb\_and\_descent » (fonction déjà existante et appliquée pour le DT)

On passe d’un calcul unique, à une boucle sur les geofence. Compte tenu de la complexité du code, cette boucle est gérée par une fonction (climb\_and\_descent\_chapeau\_chapeau qui appelle les autres fonctions avec en paramètres le nom du pit défini dans paramètres(multi-pit)).

Schéma :

La fonction climb\_and\_descent\_chapeau\_chapeau fait une boucle sur tous les noms de pit présent dans params(pit).

Pour chaque pit :

* La fonction climb\_and\_descent\_chapeau\_chapeau appelle la fonction climb\_and\_descent\_chapeau avec en paramètre le nom du pit.
* La fonction climb\_and\_descent\_chapeau calcule :
  + Altitude maximum du pit
  + Altitude minimum du pit
* La fonction climb\_and\_descent\_chapeau appelle la fonction climb\_and\_descent avec les paramètres suivants :
  + Altitude maximum du pit
  + Altitude minimum du pit
  + Treshold du pit défini à la main grâce au code C:\Users\loic\Dropbox\Jupyter\Notebooks\Eco-driving – Henri\Climb\_and\_descent\_VPS\_v2\_MODIF\_SDC.ipynb
  + Minimum distance entre les sommets défini à la main grâce au code C:\Users\loic\Dropbox\Jupyter\Notebooks\Eco-driving – Henri\Climb\_and\_descent\_VPS\_v2\_MODIF\_SDC.ipynb
  + Distance to middle : défini à la main comme rayon de la zone de scoring
  + Chosen pit : nom du pit considéré actuellement
* La fonction climb\_and\_descent calcule le score de chaque driver pour le pit en cours dans la boucle et renvoi un tableau de donnée correspondant au pit
* La fonction climb\_and\_descent\_chapeau\_chapeau concatène toutes les lignes des différents pit

Tableau intermédiaire de résultat (un tableau pour chaque pit) :



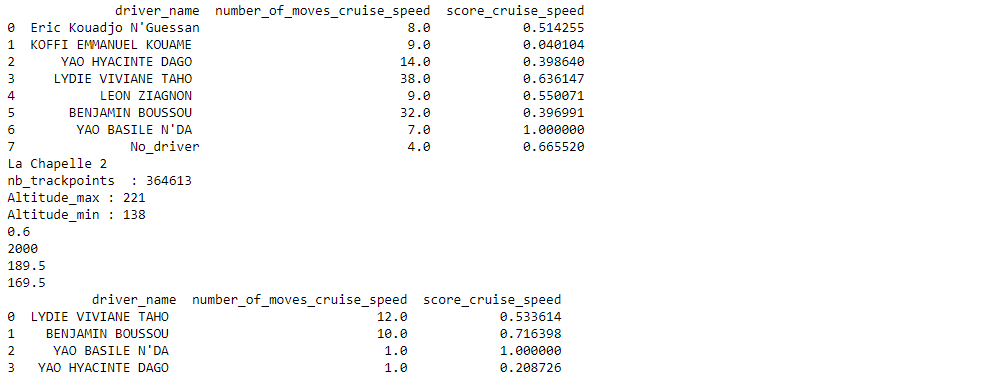
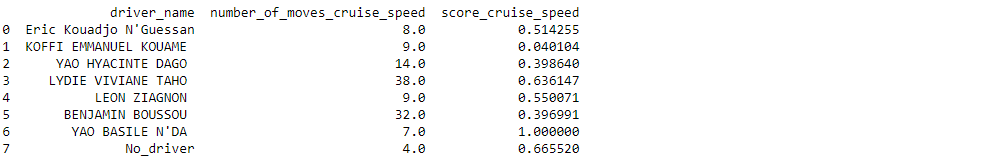
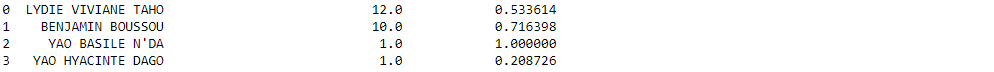


Tableau intermédiaire de résultat (toutes les lignes de toutes les geofences, plusieurs lignes par driver) :





Il reste donc la dernière étape : calculer des moyennes pondérées de score par driver.

**Recommendation pour le faire :**

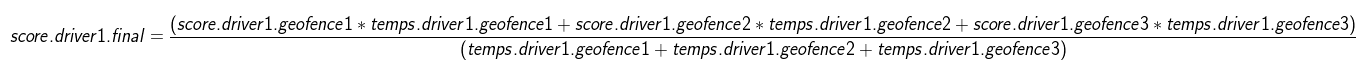
Créer un chunk de code indépendant :

Utiliser la dataframe all\_data\_geofence qui contient les informations (timeSpent et geofence) et groupby le temps par driver et par geofence afin d’obtenir un tableau de la forme :

Driver\_name / geofence / total\_time\_spent

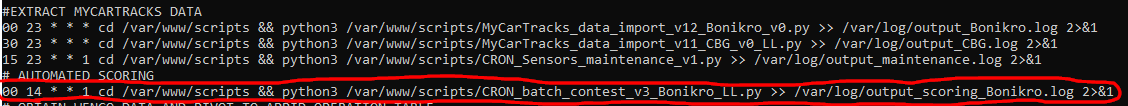
Henri / SangNord2 / 03:59:00

Ensuite croiser ce tableau avec le tableau généré par climb\_and\_descent\_chapeau\_chapeau afin d’obtenir les moyennes de score pondérées en utilisant la même formule que pour multi\_geofence :



1. **Scoring automation with the Cron Python Script *( CRON\_batch\_contest\_v3\_Bonikro\_LL.py )***
   1. Automated running with CRON

This script is automatically launched every Monday by Linux through a CRON query. By typing “crontab –l“ in Bash, you see the list of scripts that are automatically launched :



1. Inside the script

First, the script looks for the currently running contests in the MySQL tables “ecodriving\_contest” and “ecodriving\_batch\_contest”.

It extracts the corresponding *score\_type* (“DT” or “Haulage”) and the batch number, which are needed for the scoring scripts.

Then, the script calls subsequently the scripts for

1. Main scoring
2. Main continuous scoring
3. Send results

# Main scoring, then continuous scoring, then send results

*os.system("python3 Main\_scoring\_v10\_Bonikro\_LL.py "+str(contest\_id)+" "+str(batch\_number)+" "+str(batch\_duration)+" "+str(score\_type))*

*os.system("python3 Main\_continuous\_scoring\_v2\_Bonikro\_LL.py "+str(int(v["contest\_id"]))+" "+str(int(v["batch\_number"])))*

*os.system("python3 Send\_results\_scoring\_v2\_Bonikro\_LL.py "+str(int(v["contest\_id"]))+" "+str(int(v["batch\_number"]))+" "+str(int(batch\_duration)))*