Introduction  
The code is meant to be the baseline for a future driver and route analysis for Jivana Vitality. I wrote it during my internship as a project for the HSG XCamp course. While the current version is not yet capable of the in-depth analysis that is the goal for the future, it already serves as an overview for both managers and executives to see how certain routes and drivers are performing.

To add some context to the project  
Jivana Vitality India Pvt. Ltd. is an Indian for-profit company providing affordable drinking water. It was founded in 2014 by three HSG Students and has since established itself as the biggest water provider in Udaipur, India. In both rural and urban parts of India access to affordable, clean drinking water is still a big problem and many people have problems with diseases caused by polluted water. Jivana Vitality tackles this problem by providing bottled – water quality water for a fraction of the price that bottled water costs, thus making it affordable for everyone. Through five water-shops in Udaipur we deliver 80'000 liters of water daily to over 5000 active customers directly to their doorstep. With around 35 routes, each 'driven' by multiple drivers, a performance overview is getting more and more difficult. In addition Jivana is currently planning expansions to other cities, which makes a automated performance analysis a must.

## What is the point of it?

The core purpose is on one hand to grade the driver and have their bonuses depending on their grades (target=”Driver\_ID”, Output\_Ranked) and that the managers can get an overview over a specific performance. The user can execute the code from GUI.py. The process is as follows:

1. The user gives an input on over how many days back he wants to have his analysis (variable: **daysback** – accessible in GUI.py).
2. The user gives an input on how he wants his data aggregated. There are three possible aggregation levels: **Shop** (there are 5 shops), **Route** (each shop contains between 5 and 8 routes), **Driver\_ID** (each route has one main driver and one backup driver for when the main driver is sick or on leave). Also a combination of each of these three is possible as input (variable: **target** – accessible in GUI.py).  
   So when the user inputs “Driver\_ID” to target he will get a performance overview of all drivers (each driver only occurring once) but when he inputs “Route”, “Driver\_ID” he will get an overview of each drivers performance splited by route. So if a driver was driving multiple routes during the defined time frame (daysback) he will be shown multiple times, each time with the corresponding data.
3. There is a ranking logic (accessible in Ranking.py) which will determine how much bonus a driver will receive for his performance during the time frame. For this the user has to input in which percentage of performance the driver has to perform (variable: **betterScore** – accessible in GUI.py) in order to be eligible for a bonus. Based on their performance they will then get a share of the bonus-pot which is also determined by the user (variable: **Bonus** – accessible in GUI.py)

**Important**: Since I’m restructuring stuff right now everything after operations\_on\_csv will not be executed. – This means currently its only storing the downloaded queries and creating the data frames but not doing ANY analysis.

## Parts of the Code

**GUI.py**  
Will later be the graphical interface of the code, from where Main.py will be called depending on the user input. Right now its basically 50% code that I copied from sentdex (Youtube) to try out the tkinter GUI library and 50% storage room for the variables that later will come from user input in the GUI. The variables that can be changed are described in detail in the code.

**Main.py**  
This is where all the other classes and functions are called from. It consists of three parts:

1. operations\_on\_csv: Calls the stored SQL queries from QueriesSQL.py and feeds them to StoreData.py where the data will be downloaded and stored as CSV and data frame.  
2. calculations: Calls the stored data frames from StoreData and first feeds them to Analysis.py to get the Output\_Raw (basically just a data frame with ALL the information) and the secondly it feeds them to Ranking.py where specific colums will be ranked based on performance (this dataframe is then called Output\_Ranked).   
3. The created dataframes will be sent to SaveOutput, where they will be saved in CSV and HTML format.

**QueriesSQL.py**  
Is really just there to store the “text” of the SQL queries and return them to Main.py so the queries don’t clutter the rest of the code.

**StoreData.py**  
This is where the SQL queries are downloaded, saved as CSV and then transformed into data frames.

1. get\_data\_from\_SQL: Creates directories (if needed), deletes old files and stores the downloaded content from the SQL queries as CSVs.  
2.store\_csv\_to\_df: Takes the just stored CSVs and stores them as dataframes.

**Analysis.py**  
Here is where all the analysis is done. I’m loading in all the different data frames from the different SQL sources and make calculations. F.e. how many deliveries, how many trips, how many deliveries per hour, etc. In the end this will create Output\_Raw, which is used as a general performance overview with all the different stats and is also used to create the Output\_Ranked later.

**Ranking.py**  
Now that I have all the values I need in one data frame (Output\_Raw) I can take the values on which I want to base my ranking on and give each row in it a grade (columns based on the companies operational needs). This new dataframe is then stored as Output\_Ranked.

**SaveOutput.py**  
Here two things happen:

1. If possible the Output\_Raw will be split by shop. And then saved as CSVs and HTML files.  
2. Output\_Raw and Output\_Ranked are saved as a CSV and HTML file.