**RISE AGAINST HUNGER DW PROJECT**

Viktoriya Bodnar 4473240

Emre Ozden 4547248

1. **MOTIVATION AND SCOPE OF THE PROJECT**

Given two datasets (“DVD Rental” and “Rise Against Hunger”), we decided to work on the second dataset since we are interested in facing more detailed and complex analysis, and of course, because examining such a worldwide issue as hunger is extremely important nowadays.

The Date Warehousing basically exists to turn operational data into analytical information. Due to this point of view, we set our main goal of the project to *analyze who are the donors, the variation of the monetary donations with respect to time, countries and different donation levels*. Later, we will also consider some secondary questions we would like to inspect.

As a starting point in DW process we consider to follow a demand-driven approach, since we want to fulfill the user requirements to the full extent. Therefore, first of all, we simulated an interview and defined user requirements glossary, which will help us in a further work.

USER REQUIREMENT GLOSSARY

|  |  |  |  |
| --- | --- | --- | --- |
| Fact | Dimensions | Measures | History |
| Monetary Donation | Time,  Donor,  Place,  DonationLevel\* | TotalAmount,  AverageAmount(AVG) | Period from 2010 to 2016 |

(\*DonationLevel – sometimes we call it donor level or donor class.)

Then we defined a preliminary workload expressed in natural language in order to better understand what features we will take into account.

PRELIMINARY WORKLOAD

Q1 What is the total amount of givings (monetary donations) every donor (assignedto) did per year?

Q2 What is the total amount of givings per donation level per year (ordered by level,year)?

Q3 What is the relationship between every year donation and total donation amount (givings) of that donor?

Q4 How did the givings change over years (total givings grouped by year) ordered by total amount?

Q5 What are the 10 states and correspondent countries where the majority of donations are made from?

After the requirement analysis we move to the next phase such as source analysis and integration.

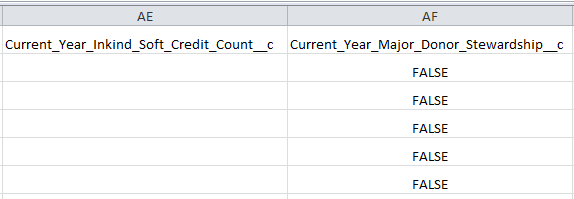
1. **OPERATIONAL DATA SOURCES INSPECTION AND PROFILING**

There are three CSV files (Accounts, Contacts, Opportunities) represented as data sources in the project. Having a look at the datasets and the data dictionary, we see that the Accounts sheet is the main source of data for us, however we also extract some relevant info from the Contacts. On the other hand, the Opportunities dataset do not provide any additional data for answering our business question.

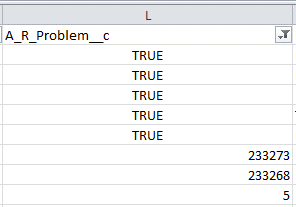
Surely, all the columns in the CSV files are not related to our project; we remove some of them and the result is nearly one fourth of the initial files. We briefly explain our reasons as follows[[1]](#footnote-1):

* We start with removing the attributes that are not related to our main goal at all. Since we will run our analysis on the fact and dimension tables, we are looking for the features that satisfy our user requirement glossary and preliminary workload. For example: “PhotoUrl” column is a very healthy column: with all distinct and readable values. But these values are URLs: not a meaningful value for our question.
* We proceed with removing all empty columns, almost empty columns, columns that have all the same values and columns where the cardinality of different values is very small. These columns, obviously, have no meaning for our analysis.

For example: “Current\_Year\_Inkind\_Soft\_Credit\_Count\_\_c” column is an all-empty column, so it does not even hold any data. Thus, this column cannot provide us any information. Next to that column we see column “Current\_Year\_Major\_Donor\_Stewardship\_\_c” which is basically having the same value for all the rows: this is an all-same column. This column can give us data, but considering the analysis,

the data will not tell us much

Column “A\_R\_Problem\_\_c” contains 233273 values, but these values are 233268 zeros and 5 other values.



As a result, these columns and similar ones are removed.

We also transformed year columns into rows using pivoting operation because in that way it is easier to manipulate and query data. Besides, we added new columns such as Average givings per account (calculated with regards to donation years and totals) and Donation level (DonorClass) that is based on yearly average givings. We assumed that there are 5 different levels of donation and divided them as follows:

|  |  |  |
| --- | --- | --- |
| DonationLevel | MinimumValue | MaximumValue |
|  |  |  |
| 1 | 0 | 500 |
| 2 | 500.001 | 1000 |
| 3 | 1000.001 | 5000 |
| 4 | 5000.001 | 10000 |
| 5 | 10000 | 99999999 |

From the Contacts table we only considered some columns related to types of accounts, address information, names assigned to accounts in order to complete our missing values and resolve some field value conflicts. Thus, our data became cleaner, more accurate and more consistent.

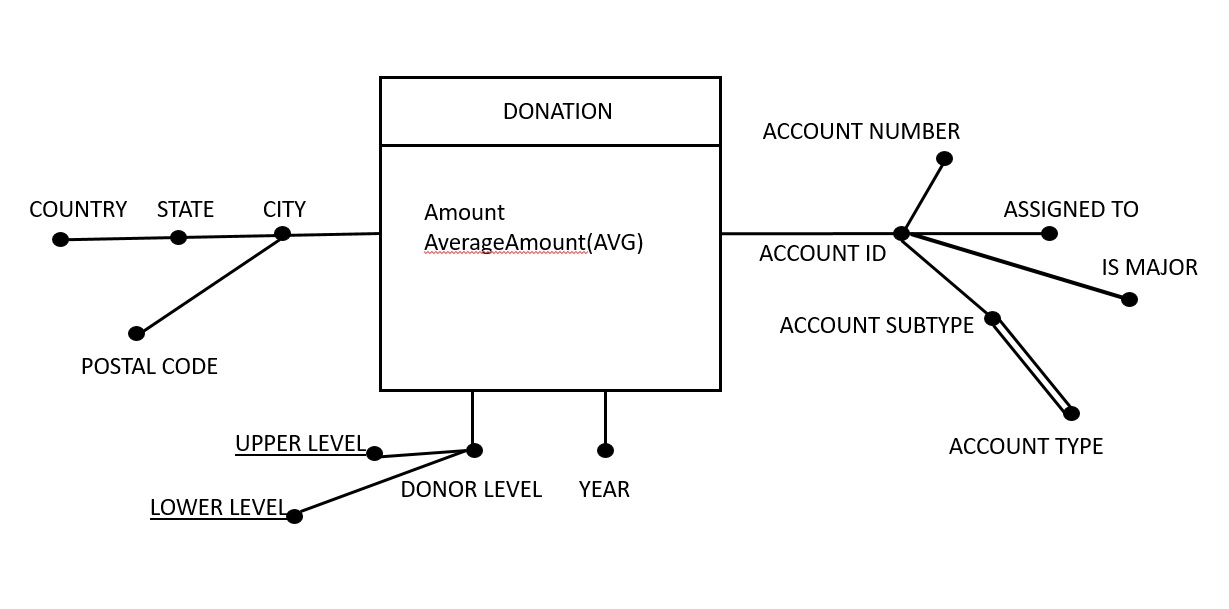
Below, there is a list of all attributes that we are interested in for our analysis.



1. **DFM**

Having considered our preliminary workload and operational resources, we defined our main fact as a donation event. Based on our used requirement glossary we would analyze every donation from different points of view. Therefore, our dimensions are: account, city, year and donor level (donation level). In the account hierarchy we noticed that the relationship between type and subtype is many-to-many, so we modelled it as a multiarch. The donor level hierarchy is composed of descriptive attributes that define the range of average amount of each level.

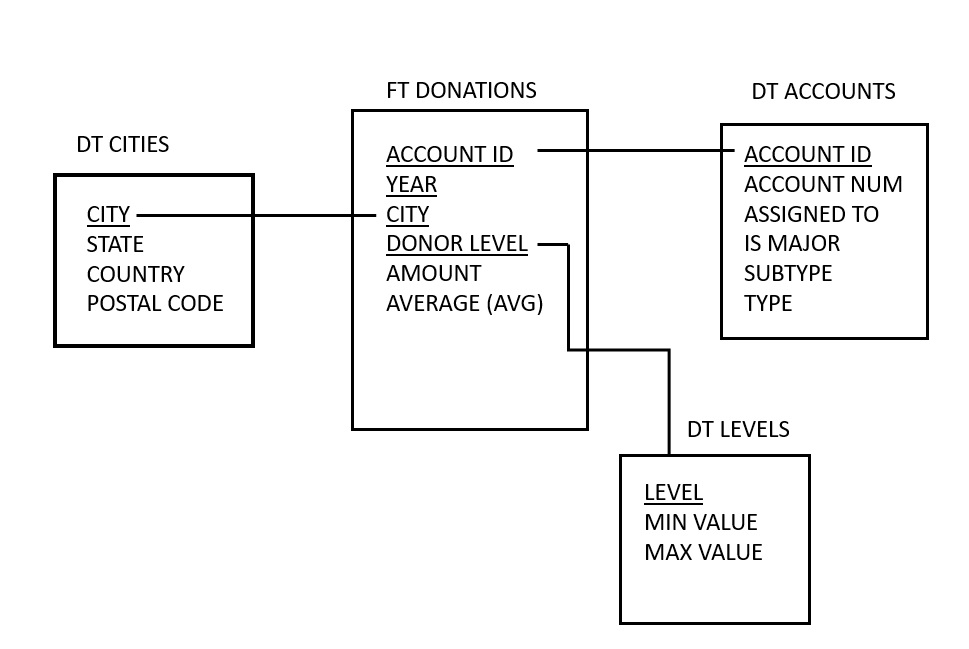
With regards to dynamicity in dimensions we considered only yearly donations between 2010 and 2016. Since no single donation information was provided, we based our analysis on that aggregation and to our mind it is better to track all the history of donations. As a result, in case the other donations take place, they are going to be inserted in the data warehouse without altering already stored data. The only thing that will be recalculated is the average of givings. The time scenario yesterday-for-today is implemented but the interval of validity is replaced by the year attribute and there is no need for master column because we will just add an account id with correspondent yearly donation.



1. **ROLLAP**

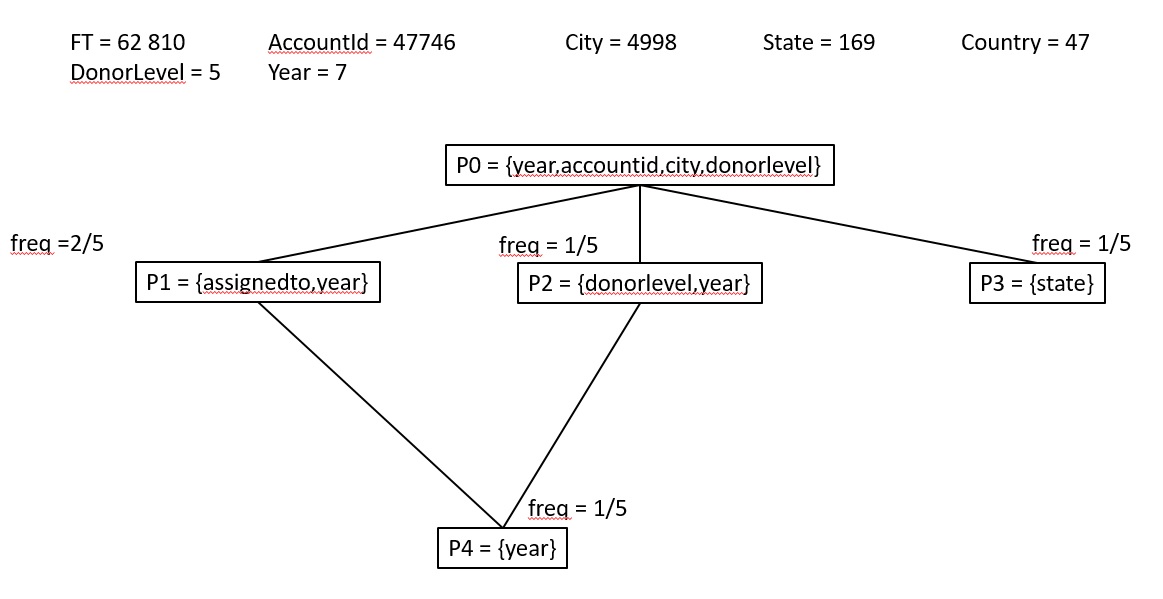
In the logical design phase, we decided to use a standard star schema, without snowflaking points. Our choice was based on some assumptions like:

* First of all, we could snowflake on state attribute to decrease the size of dimension table, but since the difference between cardinalities of city and state is small (magnitude of 10) we would better store all information together.
* Secondly, in our DFM we have a multiarch between account subtype and type. Due to the fact that the only subtype that can have different types is “other” (~250 values out of 47000) and can be seen as a default in case of missing value we are not going to create a bridge table.
* Thirdly, we created additional dimension table to outline the range of each donor level for descriptive reasons.
* Finally, we decided which views to materialize based on our data volume, attributes sizes and most frequent queries.(\*)



(\*)

In order to decide which views to materialize we draw all the aggregation patterns of our preliminary workload and take into consideration the actual sizes of tables and query frequencies. As the first step, we define the estimation of sparsity which is actual size of primary event divided by potential size = 62810/(47746x4998x5x7) ~ 7x10-6. From that number we can see that our cube is quite sparse and a lot of cells are empty. Then we calculate the ratio between P0 and P1, P2, P3. We notice that the dimensionality reduction in P1 save a lot of space and it has the highest frequency. Thus, it represents the candidate view to be stored. The other significant reduction we can see at P2 (1/47746x4998) and P3(1/47746x169x5x7). Since P4 can be retrieved from P1 or P2 there is no need to store it. In conclusion, we would materialize three views: P1, P2 and P3 in order to speed up our queries and save some relevant space.



1. **OLAP**

Sometime in order to execute complex queries and compare information at different granularity levels we need SQL OLAP extensions such as windows, frames, ranking functions and so on. We defined our workload and OLAP queries below:

**Q1. What is the total amount of givings (monetary donations) every donor(assignedto) did per year?**

select assignedto,year,givings

from accounts.nameyear

group by assignedto,year, givings

order by assignedto,year;

In this query we use our stored view nameyear, no olap operators are required.

**Q2. What is the total amount of givings per donor level per year (ordered by level,year)?**

Select donlevel,year,givings

From accounts.lvlyear

Order by donlevel,year;

In this query we also use created before view lvlyear without any olap operators.

**Q3. What is the relationship between every year donation and total donation amount (givings) of that donor?**

Select assignedto,year,givings, SUM(givings) over (partition by assignedto) as totperacc,

round(cast(givings/SUM(givings) over (partition by assignedto) as numeric), 2) as perc

From accounts.nameyear

Group by assignedto,year,givings

Order by assignedto,year;

Here, besides the fact that we use a view, we also create a window on donors in order to calculate the total amount per each donor name.

**Q4. How did the givings change over years (total givings grouped by year) ordered by total amount?**

Select year,SUM(givings ) as tot

From accounts.donations

Group by year

Order by tot desc;

In this query, we do not use either olap operators or views. Simple sql syntax with fact table is enough.

**Q5. What are the 10 states and countries where the majority of donations are made from?**

select state,country,givings

from accounts.stat

group by state,country,givings

order by givings desc

limit 10;

Here we use another view stat for retrieving states and their total givings. No olap operator is needed.

**Q6. What are the cumulative totals of the givings per account name over years?**

Select assignedto,year,givings, SUM(givings) over (partition by assignedto order by year rows unbounded preceding) as cumtot

From accounts.nameyear

Group by assignedto,year,givings

Order by assignedto,year;

In this query we use a view and create a window on donor name with all rows preceding to calculate the cumulative sum.

**Q7. What is the average of givings of current and previous year per every year per account?**

Select accountid,year,givings,AVG(givings) over (partition by accountid order by years rows 1 preceding) as mobavg

From accounts.donations

Group by accountid,year,givings

Order by accountid,year;

Here, we create a window on account id and calculate the average of givings for the current and previous year.

**Q8. What is the relationship between the average amount of givings per donor level each year and total amount over all years?**

Select donlevel,year,givings, SUM(givings) over (partition by donlevel) as totperacc,

round(cast(givings/SUM(givings) over (partition by donlevel) as numeric), 2) as perc

From accounts.lvlyear

Group by donlevel,year,givings

Order by donlevel,year;

In the query above, we use a lvlyear view and window on donor level to calculate the total amount of donations over years per donor level.

**Q9. What is the rank of accounts names due to their givings in the last (2016) year?**

Select assignedto, givings as totpername, dense\_rank() over (order by givings desc) as ranking

From accounts.nameyear

Where year = 2016

Group by assignedto,givings;

In the query above, we use a view nameyear and create a rank based on total givings.

1. **HIVE**

In order to scale out our analysis, it is not enough only one node, because very often historical data become so vast that we need to distribute it on a number of clusters. In fact, to do that we need introduce another technology, because traditional RDMS become really expensive and inefficient. Therefore, we can rely on Hive, which is a powerful infrastructure built on Hadoop that aims at analyzing data and increasing the performance by using partitions and buckets. Partitions are used to create directories in the file system and speed up the queries. Clustering is useful for even distribution based on hash keys and also to perform faster joins. As a result, we dynamically partitioned our donations table by year, clustered and sorted it by accountid into 256 buckets. The accounts table was also clustered and sorted by accountid into 256 bucket to allow the system to do a mergesort join in linear time. The cities table was partitioned by country and state in order to speed up selection queries on that fields and clustered by city into 32 buckets. In order to load data into dynamically partitioned tables we first load them into temporary tables and then overwrote the original tables with the select query from the temporary ones. OLAP queries have the same syntax, so we provide only hive workload. (for implementations see the hive document)

**H1. What is the year of donor first giving?**

Select assignedto,first\_value(year) over (partition by assignedto order by year) as firstyear

From nameyear

Group by assignedto,year

Order by assignedto;

**H2. How many there are USA donors and what is the total givings?**

select count(distinct assignedto),sum(givings) as tot

from donations,accounts,cities

where donations.accountid = accounts.accountid and donations.city = cities.city and cities.country = 'USA' group by country;

**H3. who are the major donors ?**

Select distinct assignedto from accounts where ismajor = ‘TRUE’;

**H4. who are the donors with highest level of donation and their avg givings?**

select assignedto,AVG(givings)as tot

from donations,accounts

where donations.accountid = accounts.accountid and donations.donorlevel = 5

group by assignedto

order by tot desc;

**H5. retrieve major donors, their donation levels and cumulative givings over years.**

select donations.accountid,year,donlevel, sum(givings) over (partition by donations.accountid order by year rows unbounded preceding) as cumsum

from donations,accounts

where donations.accountid = accounts.accountid and accounts.ismajor = 'TRUE'

group by donations.accountid,year,givings,donlevel

order by donations.accountid,year;

1. **SPARK SQL**

Since the Spark technology is much faster than Hive, we also considered it as a good alternative for scaling. It is easier than Spark RDD, because mainly it relies on DataFrames which are rdds with the schema on it. Due to the fact, that when manipulating DataFrames the error can be found at compilation time, we used DataFrames API to perform our analysis (main workload):

account = spark.read.option("header", "true").csv("DTAccounts.csv")

city = spark.read.option("header", "true").csv("DTCity.csv")

donation = spark.read.option("header", "true").csv("FTDonations.csv")

account.createOrReplaceTempView("accounts")

city.createOrReplaceTempView("cities")

donation.createOrReplaceTempView("donations")

Views were also created in order to speed up the queries:

lvlyear = spark.sql("SELECT donations.donorclass, donations.year,

sum(donations.givings) AS givings, avg(donations.avggivings) AS avggivings

FROM donations GROUP BY donations.donorclass, donations.year")

nameyear = spark.sql("SELECT accounts.assignedto, donations.year,

sum(donations.givings) AS givings,avg(donations.avggivings) AS avggivings

FROM donations, accounts WHERE donations.accountid = accounts.accountid

GROUP BY accounts.assignedto, donations.year")

stat = spark.sql("SELECT cities.state, cities.country,

sum(donations.givings) AS givings, avg(donations.avggivings) AS avggivings

FROM donations, cities WHERE donations.city = cities.city

GROUP BY cities.state, cities.country")

lvlyear.createOrReplaceTempView("lvlyears")

nameyear.createOrReplaceTempView("nameyears")

stat.createOrReplaceTempView("stats")

Queries in Spark using DataFrames transformations and actions:

**Q1.** nameyear.select("assignedto","year",bround("givings").alias("givings")). groupBy("assignedto","year").sum().orderBy("assignedto","year").show()

**Q2.** lvlyear.select("donorclass","year",bround("givings").alias("givings")). orderBy("donorclass","year").show()

**Q3.** nameyear.select("assignedto","year",bround("givings").alias("yearly"),

bround(sum("givings").over(Window.partitionBy("assignedto"))).alias("totperacc"),

bround(nameyear["givings"]/sum("givings").over(Window.partitionBy("assignedto")),2).alias("perc")).orderBy("assignedto").show()

**Q4.** donation.groupBy("year").agg(sum("givings").alias("tot")).orderBy("tot", ascending=False).show()

**Q5.** stat.groupBy("state","country").agg(sum("givings").alias("tot")). orderBy("tot", ascending=False).show(10)

For the Spark Workload we used another notation to demonstrate the alternative way to query in spark:

**S1 =** spark.sql("select country, sum(givings) from stats group by country order by sum(givings) desc limit 10")

**S2 =** spark.sql("select assignedto, sum(givings) from nameyears group by assignedto order by sum(givings) desc limit 10")

**S3 =** spark.sql("select donorclass, sum(givings) from lvlyears group by donorclass order by sum(givings) desc")

**S4 =** spark.sql("select city, year, count(\*) as Total\_Donation\_Count \

from donations group by city, year order by count(\*) desc limit 20")

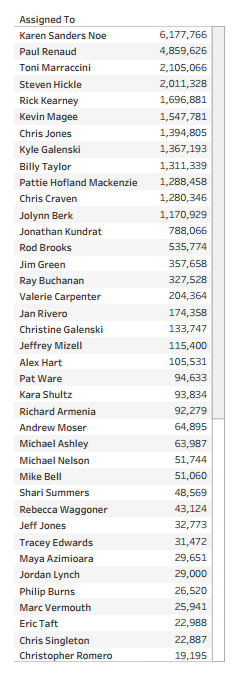
**S5 =** spark.sql("select state, country, avggivings from stats order by avggivings desc limit 10")

In order to perform transformations on dfs we need to launch the action: in our case we just show the results on the screen.

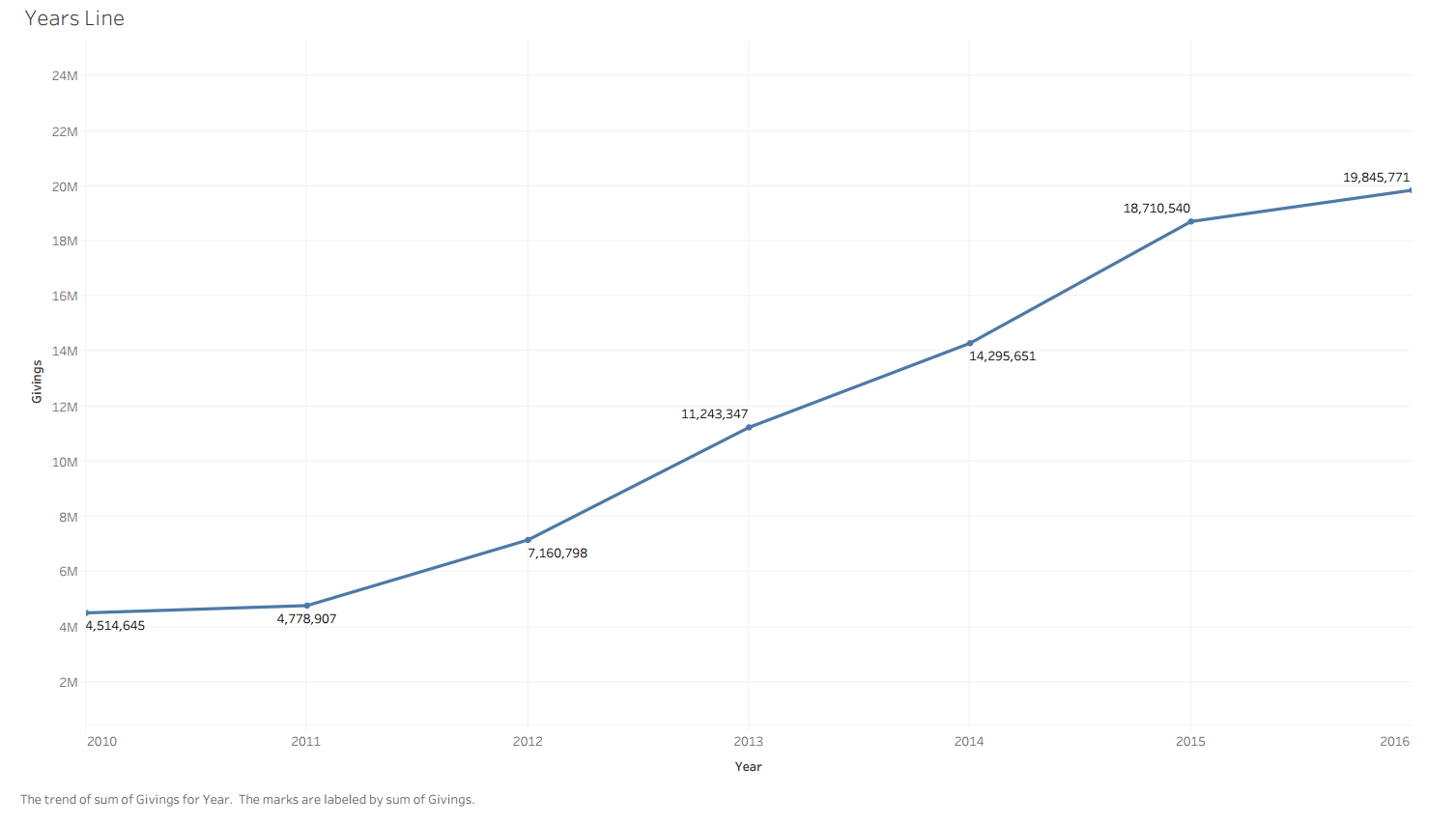
S1.show() S2.show() S3.show() S4.show() S5.show()

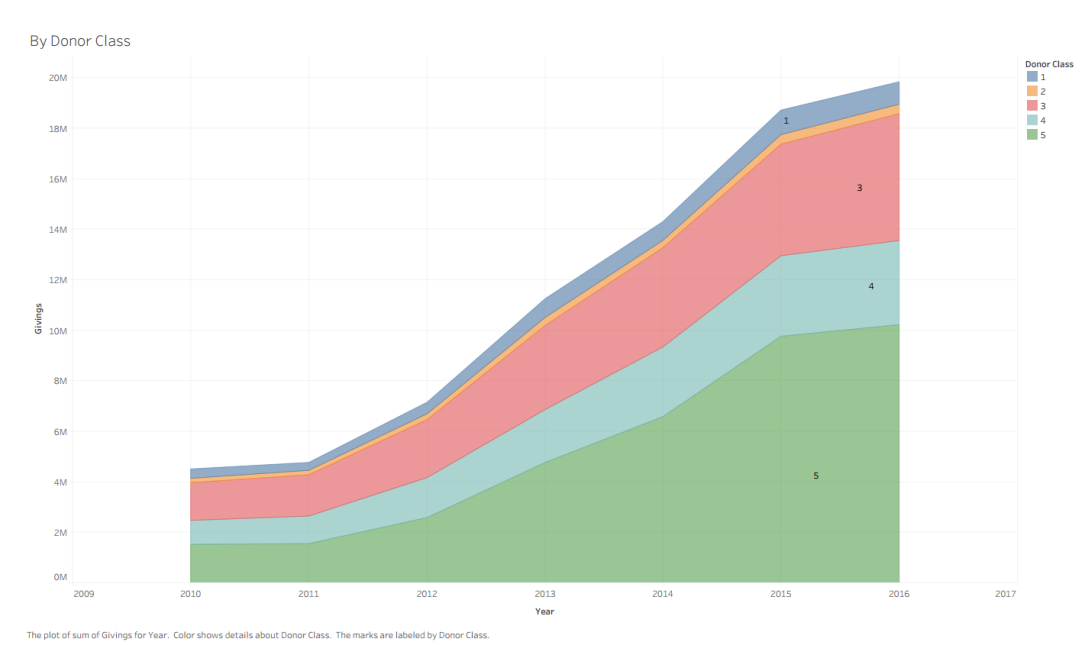
1. **TABLEAU**

The first question in which we are interested in : Who have really donated any monetary sum in the last 7 years (2010-2016)? The original list counts more than 250 people but here we show only the top donors. As we can see, the first 12 donors have donated more than 1 million of dollars to sustain the program “Rise Against Hunger”. These people`s money were registered on different accounts over the period of time from 2010 to 2016 and were transferred on behalf of different types of organizations such as Business, Congregation, Foundations, Household and so on.

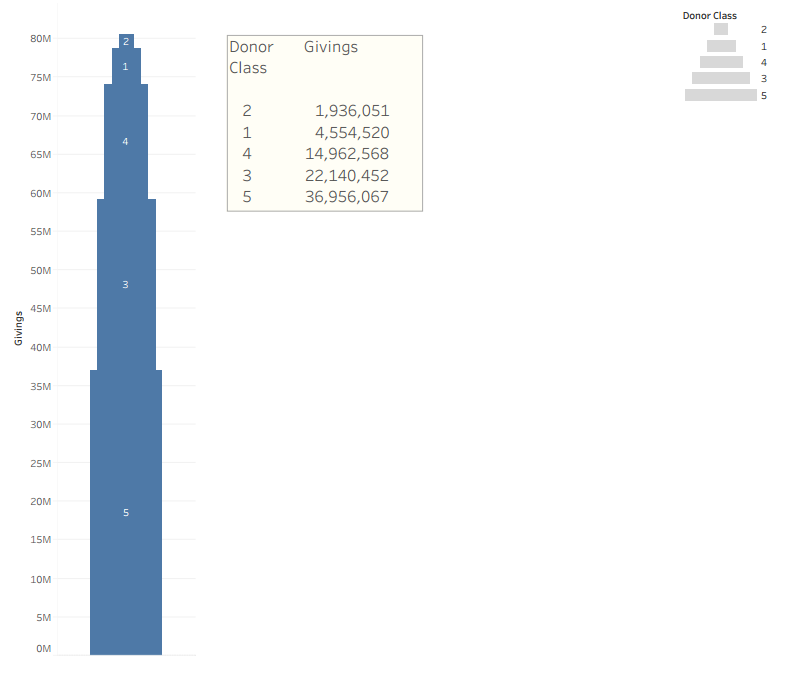
****

Having analyzed the line plot of total donations over years, we can surely say that the givings have increasing trend. From 2010 to 2016 the total sum arose from $4 million to $20 million, which is 5 times more than in the first year. This leads to confirm that more and more people support the program every year in order to fight against hunger. The most considerable increments can be seen in 2013 and 2015, as the line grows faster ( $3m more with respect to previous years).

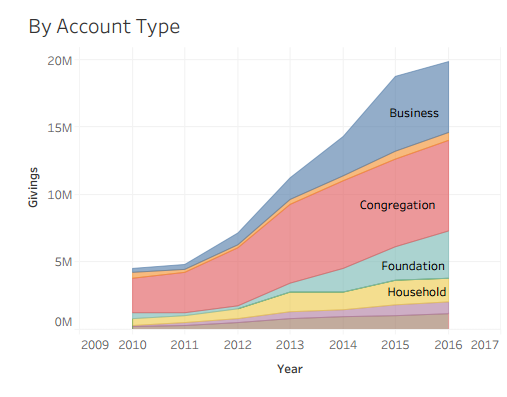
****

Furthermore, when analyzing the givings with respect to donation levels, we can observe that in general all levels are increasing over years but with different speed. For instance, the most expanding class is 5, because the total amount of donations grows from less than $2m in 2010 to more than $10m in 2016. ****

In this tower of total donations per class we can affirm that indeed the largest amount of donations are of level 5, while less money come from donations of level 2,that means that there few donors that give money of that range

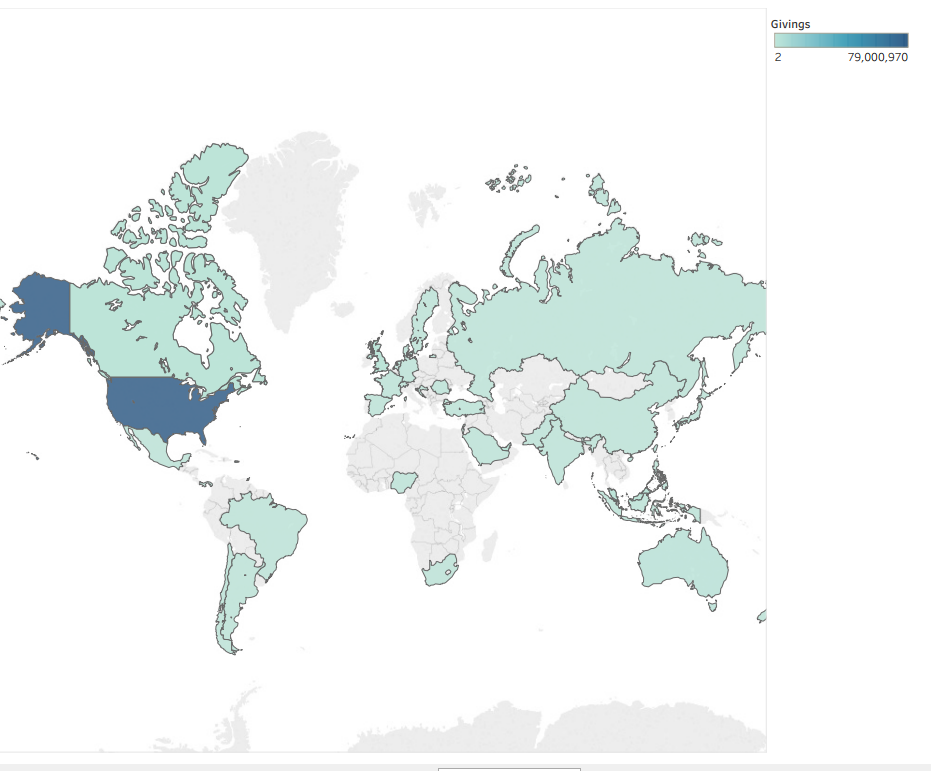
****

Due to this graph, we can see that the most amount of money comes from donations of type Congregation and Business, however the last becomes popular only after 2014 when it has a faster increment.

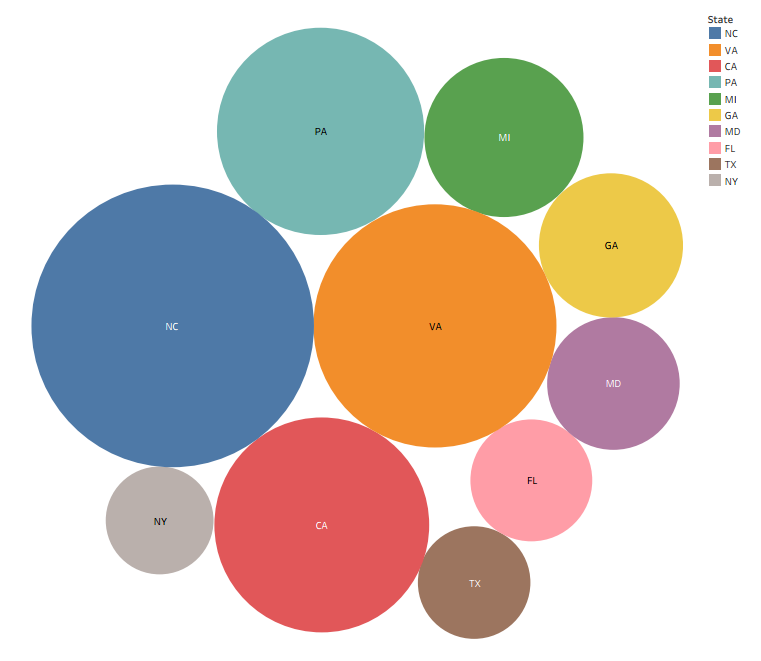
****

In order to analyze our donations with respect to space dimension, we plot some graphs based on donation origin as country, state and city.

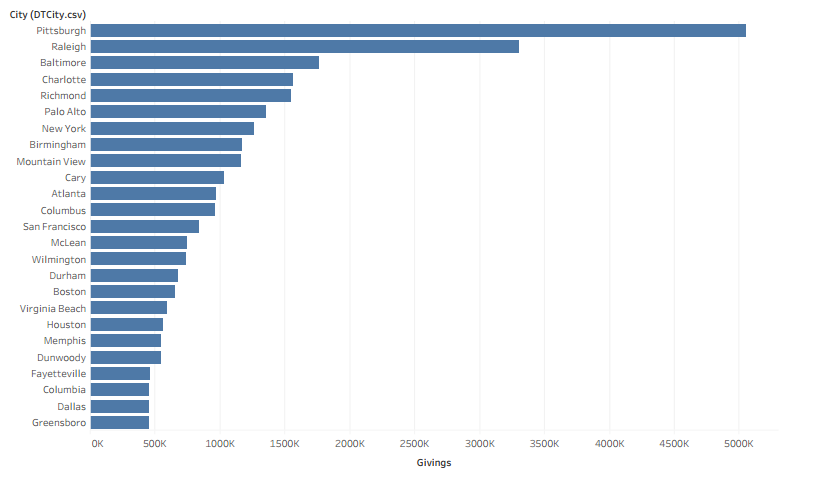
In the map below, we can notice the majority of donations come from USA (dark blue color).

****

The bubble plot shows top 10 states with the largest total sum of money given over period of interest. Thus, the North Carolina (NC), Virginia (VA) and California (CA) are the top 3 states. Looking at the other states, we can observe an important fact that the majority of the states are on the east cost of USA and represent the considerable amount of money donated.

****

On the bar graph below, we can see that the most donatable city is Pittsburgh with total donations up to 5000K dollars. The other top city is Raleigh with almost 3500K dollars. The rest varies between 0 and 1700K. This plot confirms that the East coast in average provide more money for program support.

****

To conclude, we have demonstrated that more and more people became interested in supporting such an important program as “Rise Against Hunger”. We saw that the numbers of donors just like the number of donations increased over years and continue to increase nowadays. The most supporting country is USA and moreover, the majority of donations are concentrated on the east coast. All of this can be taken into consideration in order to expand the program all over the world and to encourage even more people to donate, since to fight against this problem we have to be unite.

1. **RESOURCES**

<http://www.riseagainsthunger.org/>

<https://it.wikipedia.org/wiki/Rise_Against_Hunger>

<https://docs.docker.com/>

<https://docs.trifacta.com/>

<https://cwiki.apache.org/confluence/display/Hive/LanguageManual>

<https://spark.apache.org/docs/latest/sql-programming-guide.html>

1. The explanations are based on Accounts.csv file. [↑](#footnote-ref-1)