**John Bartlett**

**IST 659**

**Project**

**Personal Life Tracking**

**&**

**Dynamic Habit Discovery**

**Summary:**

Personal Life tracking, there are many applications today that track various aspects of one’s day to day life. From financial planning apps to calorie counters to exercise trackers, if you can think of it then it probably already exists. There are a few apps that I use to track my activities, Samsung Health, Chase Bank, Google App Suite like Calendar and Maps. My idea is to combine the data from as many of these apps as possible into one universal database to visualize the data to find interesting correlations between them. This is a lofty idea so I will be starting from the ground up adding data on a by application basis, starting with Samsung health data. I will use already existing data on diet, exercise, sleep time, blood pressure, etc. Later I’d like to add the data from my credit card, to correlate my credit card use with exercise and eating habits. My reach goal will be to combine more complicated data like my GPS data from google maps or even google search history (oof). The way that my use of this data will differ from its intended purpose is that I will use it in a more self-reflective and productive. I’d like to gain a more refined understanding of my habits.

The method I use to extract the data will also govern how I structure it. Due to the daunting task of manually editing all the data I have, the columns and attributes I use will be mostly governed by what the applications provide. That leads me into my methods, I will explain those next.

**Summary Addendum:**

The core of my project is around automating the recreation of Data from frequented apps within a MSSQL database of which I have complete control. I wanted to focus on applications that tell me about myself, and to be able to visualize that data conveniently, to look for correlations and to track progress in certain life changes I might make, like training to run a marathon, or trying to put more into my savings.

**Methods:**

You may be asking, ‘how are you going to extract all this data!? well many of the apps I’ve mentioned have methods of getting your data into a single file. This data is easily extractable, and quite uniform, therefore easily processed. In my past I have built Python scripts that take plain text files, csv’s etc. and delimit them into a sequence of SQL insert statements that I can then insert into my database. This process will fast track the data actual data creation in my database. Also because I am using existing data extracted from external sources this will mean that most of my attributes will be determined by the ways in which my extracted data is constructed.

As a specific example, Samsung Health delivers user data in csv files. So, meals and food items along with their respective calories are listed in the convenient csv format, like so:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| max\_rpm | live\_data | mean\_rpm | distance | max\_caloricburn\_rate | additional | altitude\_gain | deviceuuid | duration | update\_time | datauuid | location\_data | decline\_distance | altitude\_loss | mean\_caloricburn\_rate | incline\_distance | max\_altitude | max\_cadence | create\_time | time\_offset | mean\_speed | mean\_heart\_rate | custom | count\_type | min\_altitude | count | end\_time | max\_speed | exercise\_type | start\_time | max\_power | pkg\_name | mean\_power | min\_heart\_rate | calorie | comment | max\_heart\_rate | mean\_cadence | exercise\_custom\_type | |
|  | f9f0bff2-af33-41a4-86be-3368bac20b39.live\_data.json | | 1455 |  |  |  | bO6YcZMOlP | 1058805 | 22:46.4 | f9f0bff2-af33-41a4-86be-3368bac20b39 | | | |  |  |  |  | 22:46.4 | UTC-0300 | 1.374191 |  |  |  |  |  | 14:03.8 |  | 1001 | 56:25.0 |  | com.sec.android.app.shealth | | | 75.84 |  |  |  |  |  |

As you can see the above table fly’s off the page. This will not be an issue as most of this data is empty, or only useful to the app. You can see distance and duration and update time in the above table, these are some key points I will be interested in.

Another example is the Chase credit card data, this data is delivered right in my monthly statement. The monthly statement is a searchable pdf, meaning you can conveniently copy and past into plain text, see the example below:

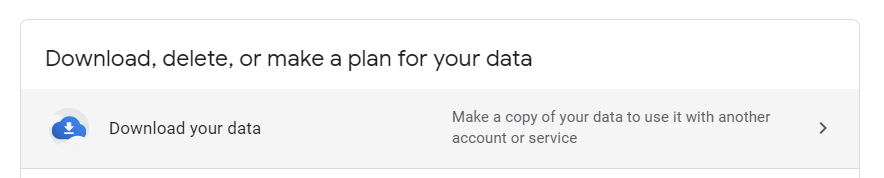
12/03 New York Style Pizza SYRACUSE NY 7.00

12/04 WEGMANS #030 FAYETTEVILLE NY 22.55

12/06 PRESSROOM PUB SYRACUSE NY 35.11

This data will be easier to process into insert statements, as there isn’t much there and what’s there is quite simple.

Google is where things become a stretch, there is a large amount of data you can extract via your google account, there is a link on your google account page that looks like this:



This aspect will be a bit more complicated as data is extracted in few different formats: .json .txt .icalendar .csv .html etc.. Each of these aspects will require a unique solution for adapting the data to a sql database. This will pose a few new challenges, but there is one upside: the google data downloader has an auto export function. This means that the entire process of this Google export can be fully automated (yay). This will be a great feature if I am to maintain this database after its use in this class which is something I hope to do. My initial targets with the google aspect of my database will be search history, browser history, google maps data and lastly google assistant voice searches.

**Methods Addendum:**

As you can see from my Python scripts below, my methods were accurately predicted. I was able to find a way to export my Chase data into a .csv, making it far easier to generate inserts using the csv python module. The method of creating this database goes as follows:

1. Export data from Samsung health and chase
2. Change the file parameters within my Pyhton scripts to point to the csv’s you are delimiting.
3. Run the delimiting scripts
4. Open the output .sql scripts
5. Run the DDL to recreate the database
6. Run the FOOD insert script first, then immediately run the Clean\_Food stored procedure (this will allow Meals to be correctly inserted)
7. Then run the rest of the insert scripts in any order
8. Lastly run Sort\_Credit\_Misc and Sort\_Checking\_Misc to sort Chase data into my added categories
9. All set!

If you follow these steps, anyone Using Chase or Samsung health could achieve what this project set out to do.

**Stakeholders:**

Me, myself and I.

**Business Rules:**

Data normalization will be key for this project, when dealing with nutrition tracking or exercise tracking there are many factors that could throw of the clarity of the database. One of the most important aspects of my database will be to track this broad swath of data over one universal standard, that standard will of course be Date/Time. Normalization of my datetime format will be pivotal to the success of my project.

Meals will be made up of different combinations of FoodIDs, therefore, to normalize the Meals table and reduce duplicates there will need to be an intermediary table to facilitate multiple combinations of food items making up one meal.

Health information will have separate tables for each item in said category. So, Pulse, Steps, Exercise will all be separate. Many things will be derived from other information, for instance, exercise duration will be determined from the start and end time. Getting a spotlight health summary will also be possible, this summary will combine several derived attributes: Calories Consumed, derived from the meals table (all the meal calories added up for the day), Calories Burned, derived from the aggregate of calories burned from exercise, plus the standard amount burned by just living (which will be determined by weight and some close data tracking) And calories burned per step.

Expenses should contain info about each transaction and be categorized into single categories. There will then be a Financial Summary table that will derive the total amount of money spent per day by aggregating all of the expenses from the expenses table.

Searches will be a clearer data and as of, yet I cannot think of any ancillary tables or business rules to build around this table.

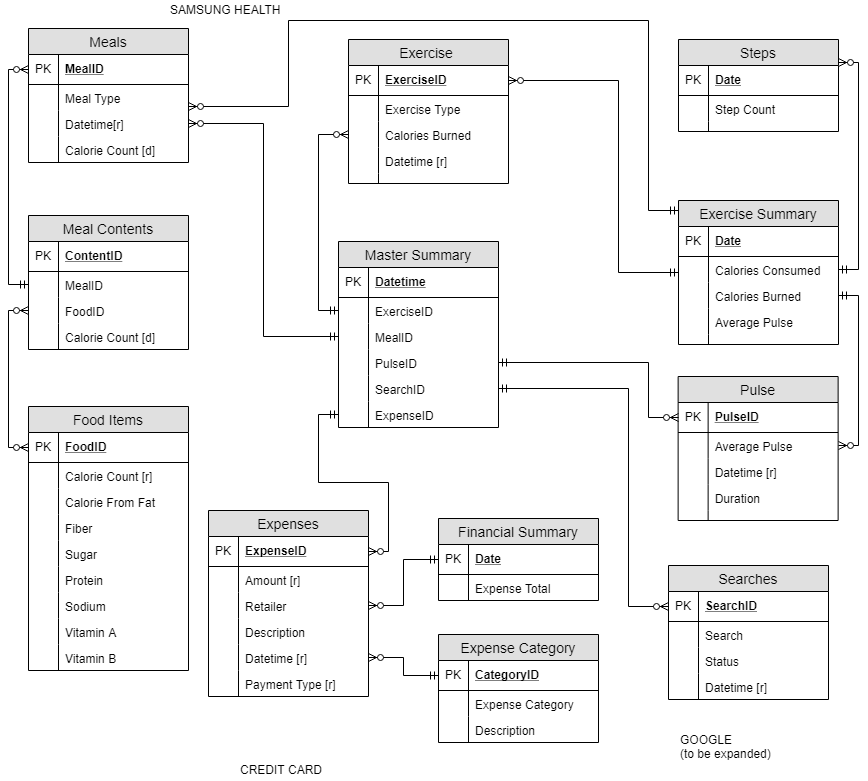
For more information please see my models below:

**Business Rules Addendum:**

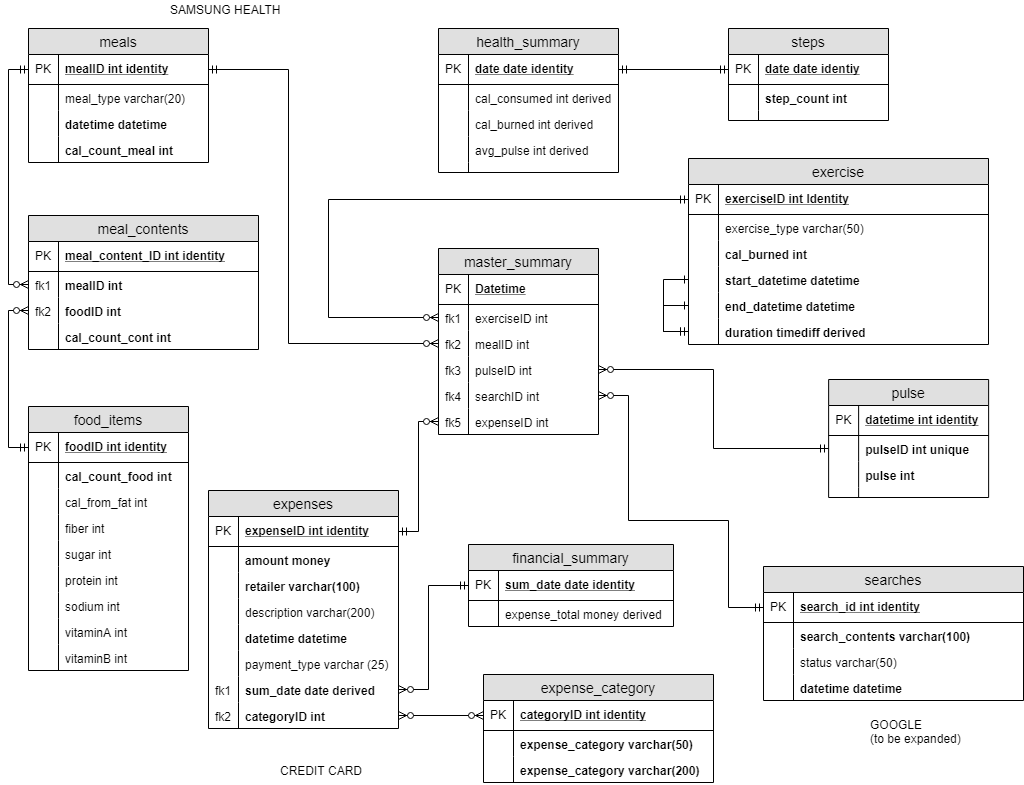
I abandoned my universal standard of using Datetime as a primary key for my database. Headache over scripting inserts with datetimes of universal formats and worrying about having vast amount of NULL data that points to times with no data caused me to make this decision.

Also using Power BI reduced the need for a datetime Primary key. And completely removed the need to create summary tables that would have combined data from separate tables. Power BI can do this nicely.

**Conceptual Model**



**Logical Model**



**Personal Life Tracking**

**&**

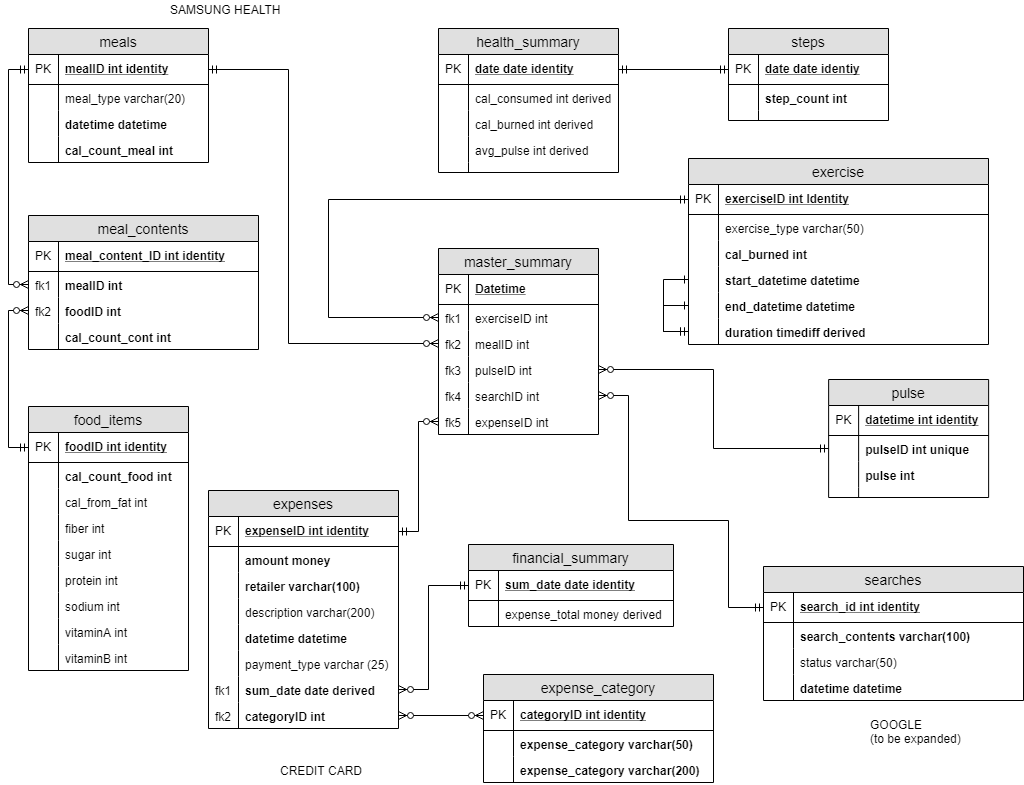
**Dynamic Habit Discovery**

**PART 2**

**PART II**

**A few Changes**

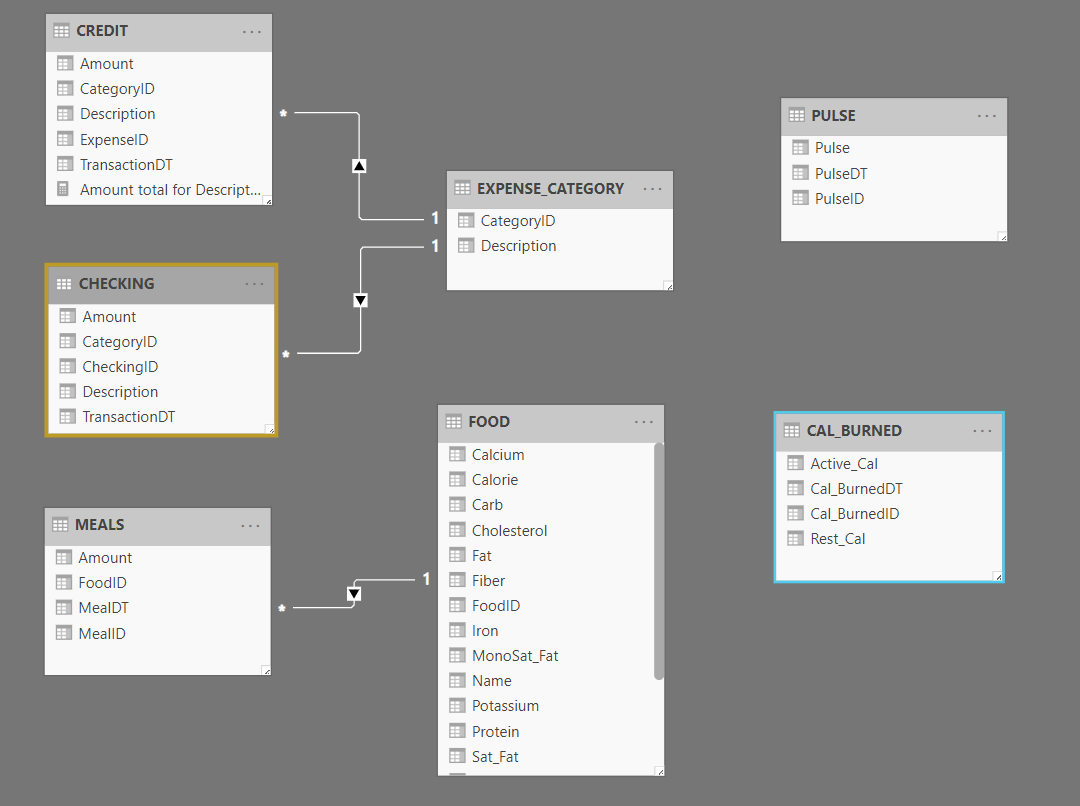
I’ve deviated from my original database concepts in more than a few ways. Ill comment on my original logical model for context:



1. The meals and meal\_contents tables have been combined into one table.
2. Most of the ‘Summary’ Tables have been removed and will be replace via Power BI diagrams.
3. Expenses have been separated into 2 separate tables, **Checking and Credit**.
4. The google searches table has been pushed back, the python export script proved difficult to design and still doesn’t work, the data is delivered in an HTML all on one line, which is much more difficult to delimit.

**Conceptual Model from Power BI**

I’ve utilized Power BI Modeling feature to generate a pseudo conceptual model that is directly corelated with my actual database, it has been greatly reduced in complexity when compared to my original model.



With all that being said, my DDL is below.

**DDL**

------------------------------------------------------------------------------  
  
--Table creates for Samsung Health data  
------------------------------------------------------------------------------  
  
DROP TABLE IF EXISTS dbo.PULSE;  
GO  
  
create table PULSE  
(  
 PulseID int identity  
 constraint PULSE\_pk  
 primary key nonclustered,  
 PulseDT datetime,  
 Pulse int  
)  
go  
  
-----------------------------------------------------------------------------------------------------  
  
DROP TABLE IF EXISTS dbo.CAL\_BURNED;  
GO  
  
create table CAL\_BURNED  
(  
 Cal\_BurnedID int identity  
 constraint CAL\_BURNED\_pk  
 primary key nonclustered,  
 Cal\_BurnedDT datetime,  
 Rest\_Cal int,  
 Active\_Cal int  
)  
go  
  
-----------------------------------------------------------------------------------------------------  
  
DROP TABLE IF EXISTS dbo.STEPS;  
GO  
  
create table STEPS  
(  
 StepsID int identity  
 constraint STEPS\_pk  
 primary key nonclustered,  
 StepsDT datetime,  
 Steps\_Count int,  
 Speed float,  
 Cal\_Burned float  
)  
go  
  
-----------------------------------------------------------------------------------------------------  
  
alter table MEALS  
 drop constraint MEALS\_FOOD\_FoodID\_fk  
go  
  
DROP TABLE IF EXISTS dbo.FOOD;  
GO  
  
create table FOOD  
(  
 FoodID int identity not null  
 constraint FOOD\_pk  
 primary key nonclustered,  
 Name varchar(100) not null,  
 Serving\_Info varchar(100),  
 Calorie float default 0,  
 Carb float default 0,  
 Fat float default 0,  
 Protein float default 0,  
 Fiber float default 0,  
 Cholesterol float default 0,  
 VA float default 0,  
 Calcium float default 0,  
 VC float default 0,  
 Sat\_Fat float default 0,  
 MonoSat\_Fat float default 0,  
 Potassium float default 0,  
 Sodium float default 0,  
 Sugars float default 0,  
 Iron float default 0  
)  
go  
  
create unique index FOOD\_FOODID\_uindex  
 on FOOD (FoodID)  
go  
  
-----------------------------------------------------------------------------------------------------  
  
DROP TABLE IF EXISTS dbo.MEALS;  
GO  
  
create table MEALS  
(  
 MealID int identity,  
 FoodID int not null  
 constraint MEALS\_FOOD\_FoodID\_fk --fk linking meals to foods  
 references FOOD,  
 MealDT datetime,  
 Amount float  
)  
go  
  
create unique index MEALS\_MealID\_uindex  
 on MEALS (MealID)  
go  
  
alter table MEALS  
 add constraint MEALS\_pk  
 primary key nonclustered (MealID)  
go  
  
  
------------------------------------------------------------------------------  
  
-- Table creates for finacial data  
  
------------------------------------------------------------------------------  
  
alter table CREDIT  
 drop constraint CREDIT\_CATEGORY\_CategoryID\_fk  
go  
  
alter table CHECKING  
 drop constraint CHECKING\_CAT\_FK  
go  
  
DROP TABLE IF EXISTS dbo.EXPENSE\_CATEGORY;  
GO  
  
create table EXPENSE\_CATEGORY  
(  
 CategoryID int identity  
 constraint EXPENSE\_CATEGORY\_pk  
 primary key nonclustered,  
 Description varchar(100)  
)  
go  
  
create unique index EXPENSE\_CATEGORY\_CATEGORYID\_uindex  
 on EXPENSE\_CATEGORY (CategoryID)  
go  
  
  
--Inserts for categorizations. Expanding on the shopping category and creating a new category payments for tracking card payments  
INSERT into EXPENSE\_CATEGORY(Description) VALUES('Misc');  
INSERT into EXPENSE\_CATEGORY(Description) VALUES('Payments');  
INSERT into EXPENSE\_CATEGORY(Description) VALUES('Automotive');  
INSERT into EXPENSE\_CATEGORY(Description) VALUES('Bills & Utilities');  
INSERT into EXPENSE\_CATEGORY(Description) VALUES('Education');  
INSERT into EXPENSE\_CATEGORY(Description) VALUES('Entertainment');  
INSERT into EXPENSE\_CATEGORY(Description) VALUES('Food & Drink');  
INSERT into EXPENSE\_CATEGORY(Description) VALUES('Gas');  
INSERT into EXPENSE\_CATEGORY(Description) VALUES('Gifts & Donations');  
INSERT into EXPENSE\_CATEGORY(Description) VALUES('Groceries');  
INSERT into EXPENSE\_CATEGORY(Description) VALUES('Health & Wellness'); --medical  
INSERT into EXPENSE\_CATEGORY(Description) VALUES('Home');  
INSERT into EXPENSE\_CATEGORY(Description) VALUES('Personal');  
INSERT into EXPENSE\_CATEGORY(Description) VALUES('Amenities');  
INSERT into EXPENSE\_CATEGORY(Description) VALUES('Clothing');  
INSERT into EXPENSE\_CATEGORY(Description) VALUES('Travel');  
INSERT into EXPENSE\_CATEGORY(Description) VALUES('Rent');  
INSERT into EXPENSE\_CATEGORY(Description) VALUES('Wages');  
INSERT into EXPENSE\_CATEGORY(Description) VALUES('ATM');  
INSERT into EXPENSE\_CATEGORY(Description) VALUES('Professional Services');  
  
  
-----------------------------------------------------------------------------------------------------  
  
DROP TABLE IF EXISTS dbo.CREDIT;  
GO  
  
create table CREDIT  
(  
 ExpenseID int identity  
 constraint CREDIT\_pk  
 primary key nonclustered,  
 TransactionDT datetime,  
 Description varchar(100),  
 CategoryID int not null default 0  
 constraint CREDIT\_CATEGORY\_CategoryID\_fk -- fk linking credit to its categorys  
 references EXPENSE\_CATEGORY,  
 Amount float default 0  
)  
go  
  
-----------------------------------------------------------------------------------------------------  
  
DROP TABLE IF EXISTS dbo.CHECKING;  
GO  
  
create table CHECKING  
(  
 CheckingID int identity  
 constraint CHECKING\_pk  
 primary key nonclustered,  
 TransactionDT datetime,  
 Description varchar(100),  
 Amount float default 0  
)  
go  
  
ALTER table CHECKING  
 add CategoryID int not null default 1;  
  
  
ALTER table CHECKING  
 add CONSTRAINT CHECKING\_CAT\_FK --add fk for checking to use the category table  
 FOREIGN KEY (CategoryID)  
 references EXPENSE\_CATEGORY(CategoryID);  
  
-----------------------------------------------------------------------------------------------------  
  
-- VIEWS  
  
-----------------------------------------------------------------------------------------------------  
DROP VIEW IF EXISTS dbo.Meal\_Contents;  
DROP VIEW IF EXISTS dbo.Credit\_Balance;  
DROP VIEW IF EXISTS dbo.Credit\_History;  
DROP VIEW IF EXISTS dbo.Checking\_Balance;  
go;  
  
CREATE VIEW Meal\_Contents as  
SELECT MealDT, F.Name, Amount, F.Serving\_Info, Calorie, Carb, Fat, Protein, Fiber from MEALS --view for looking at meals with their food description  
JOIN FOOD F on MEALS.FoodID = F.FoodID  
go;  
CREATE VIEW Credit\_History as  
SELECT TransactionDT, CREDIT.Description as Description, EC.Description as Category, Amount --view for looking at credit history joined with categorys  
from CREDIT  
join EXPENSE\_CATEGORY EC on CREDIT.CategoryID = EC.CategoryID  
go;  
CREATE VIEW Credit\_Balance as  
SELECT *ROUND*(*SUM*(Amount), 2) as Balance  
FROM CREDIT  
go;  
CREATE VIEW Checking\_Balance as  
SELECT *ROUND*(*SUM*(Amount), 2) AS Balance  
FROM CHECKING  
go;  
  
-----------------------------------------------------------------------------------------------------  
  
-- Procedures  
  
-----------------------------------------------------------------------------------------------------  
  
DROP procedure IF EXISTS *Sort\_Credit\_Misc*;  
DROP procedure IF EXISTS *Sort\_Checking\_Misc*;  
go;  
  
  
Create procedure *Sort\_Credit\_Misc* --adds my custom categorys to credit  
as  
 begin  
 UPDATE CREDIT SET CategoryID = 14 WHERE Description LIKE 'PAYPAL%'; -- SORT INTO AMENITYS  
 UPDATE CREDIT SET CategoryID = 13 WHERE Description LIKE '%UNIQLO%' OR Description LIKE '%SHOES%' OR  
 Description LIKE '%MASSDROP%' OR Description LIKE '%CLOTHES%'; --SORT INTO CLOTHES  
 end  
go;  
  
  
Create procedure *Sort\_Checking\_Misc* --adds my custom categorys to checking  
as  
 begin  
 UPDATE CHECKING SET CategoryID = 22 WHERE Description LIKE '%ATM%' OR Description LIKE '%Withdrawal%'; -- SORT INTO AMENITYS  
 UPDATE CHECKING SET CategoryID = 20 WHERE Description LIKE '%Premier%' or Amount = -850;  
 UPDATE CHECKING SET CategoryID = 21 WHERE Description LIKE '%DIR DEP%' OR Description LIKE '%PAYROLL%';  
 UPDATE CHECKING SET CategoryID = 2 WHERE Description LIKE '%Payment%';  
 end  
go;  
  
create procedure *Clean\_Food* -- removes food entries with duplicate names  
as  
 begin  
 delete from FOOD where FoodID not in(select *min*(FoodID) as id from food group by Name)  
 end

**I then wrote a few python procedures that will delimit the data I source from Chase and the Samsung Health app.**

**PYTHON:**

import csv  
  
  
  
def delimit\_credit():  
 with open('E:\OneDrive\DataScience\IST 659\Project\Chase\Credit2019.csv') as csv\_file:  
 forecast\_reader = csv.reader(csv\_file, delimiter=',')  
  
  
 ##initial pointers  
 count = 0  
  
 ##Full file read  
 for row in forecast\_reader:  
 if count == 0:  
 outPut = open('E:\OneDrive\DataScience\IST 659\Project\\OutPut\\Credit2019.sql' , 'w+')  
 count += 1  
 continue  
 elif (row[3] == 'Shopping'):  
 if row[2].find("'") != -1:  
 row[2] = row[2].replace("'", "''")  
 outPut.write('INSERT into CREDIT(TransactionDT, Description, CategoryID, Amount) VALUES (\'' + row[0]  
 + '\', \'' + row[2] + '\', 1,' + row[5] + ')\n')  
 continue  
 elif (row[3] == '' or row[3] is None):  
 if row[2].find("'") != -1:  
 row[2] = row[2].replace("'", "''")  
 outPut.write('INSERT into CREDIT(TransactionDT, Description, CategoryID, Amount) VALUES (\'' + row[0]  
 + '\', \'' + row[2] + '\', 2,' + row[5] + ')\n')  
 continue  
 else:  
 if row[2].find("'") != -1:  
 row[2] = row[2].replace("'", "''")  
 outPut.write('INSERT into CREDIT(TransactionDT, Description, CategoryID, Amount) VALUES (\'' + row[0]  
 + '\', \'' + row[2] + '\', (SELECT(CategoryID) FROM EXPENSE\_CATEGORY where Description = \''  
 + row[3] + '\' ), ' + row[5] + ')\n')  
  
  
def delimit\_checking():  
 with open('E:\OneDrive\DataScience\IST 659\Project\Chase\\Checking.csv') as csv\_file:  
 forecast\_reader = csv.reader(csv\_file, delimiter=',')  
  
 ##initial pointers  
 count = 0  
  
 ##Full file read  
 for row in forecast\_reader:  
 if count == 0:  
 outPut = open('E:\OneDrive\DataScience\IST 659\Project\OutPut\\Checking.sql' , 'w+')  
 count += 1  
 continue  
 else:  
 if row[2].find("'") != -1:  
 row[2] = row[2].replace("'", "''")  
 outPut.write('INSERT into CHECKING(TransactionDT, Description, Amount) VALUES (\'' + row[1]  
 + '\', \'' + row[2] + '\',' + row[3] + ')\n')

def delimit\_pulse():  
 with open('E:\OneDrive\DataScience\IST 659\Project\SamsungHealth\pulse.csv') as csv\_file:  
 forecast\_reader = csv.reader(csv\_file, delimiter=',')  
  
 ##initial pointers  
 count = 0  
  
 ##Full file read  
 for row in forecast\_reader:  
 if (count == 0):  
 outPut = open('E:\OneDrive\DataScience\IST 659\Project\OutPut\\' + row[0] + '.sql' , 'w+')  
 count += 1  
 continue  
 elif (count == 1):  
 count += 1  
 continue  
 else:  
 outPut.write('insert into PULSE (PulseDT, Pulse) VALUES (\'' + row[0] + '\', ' + row[6] +')\n')  
  
##read forecast  
def delimit\_cal\_burned():  
 with open('E:\OneDrive\DataScience\IST 659\Project\SamsungHealth\cal\_burned.csv') as csv\_file:  
 forecast\_reader = csv.reader(csv\_file, delimiter=',')  
  
 ##initial pointers  
 count = 0  
  
 ##Full file read  
 for row in forecast\_reader:  
 if (count == 0):  
 outPut = open('E:\OneDrive\DataScience\IST 659\Project\OutPut\\' + row[0] + '.sql' , 'w+')  
 count += 1  
 continue  
 elif (count == 1):  
 count += 1  
 continue  
 else:  
 outPut.write('insert into CAL\_BURNED (Cal\_BurnedDT, Rest\_Cal, Active\_Cal) VALUES ' +  
 ' (\'' + row[3] + '\', ' + row[6] + ', ' + row[7] + ')\n')  
  
  
def delimit\_steps():  
 with open('E:\OneDrive\DataScience\IST 659\Project\SamsungHealth\steps.csv') as csv\_file:  
 forecast\_reader = csv.reader(csv\_file, delimiter=',')  
  
 ##initial pointers  
 count = 0  
  
 ##Full file read  
 for row in forecast\_reader:  
 if (count == 0):  
 outPut = open('E:\OneDrive\DataScience\IST 659\Project\OutPut\\' + row[0] + '.sql' , 'w+')  
 count += 1  
 continue  
 elif (count == 1):  
 count += 1  
 continue  
 else:  
 outPut.write('insert into STEPS(StepsDT, Steps\_Count, Speed, Cal\_Burned) VALUES ' +  
 ' (\'' + row[0] + '\', ' + row[4] + ', ' + row[6] + ', ' + row[12] + ')\n')  
  
  
def delimit\_food():  
 with open('E:\OneDrive\DataScience\IST 659\Project\SamsungHealth\\food\_info.csv') as csv\_file:  
 forecast\_reader = csv.reader(csv\_file, delimiter=',')  
  
 ##initial pointers  
 count = 0  
  
 ##Full file read  
 for row in forecast\_reader:  
 if (count == 0):  
 outPut = open('E:\OneDrive\DataScience\IST 659\Project\OutPut\\' + row[0] + '.sql' , 'w+')  
 count += 1  
 continue  
 elif (count == 1):  
 count += 1  
 continue  
 elif row[27].find("'") != -1:  
 row[27] = row[27].replace("'", "''")  
 x = 31  
 while x > 0:  
 if row[x] is None or row[x] == '':  
 row[x] = '0'  
 x -= 1  
 outPut.write('insert into FOOD(Name, Serving\_Info, Calorie, Carb, Fat, Protein, Fiber, ' +  
 'Cholesterol, VA, Calcium, VC, Sat\_Fat, MonoSat\_Fat, Potassium, Sodium, Sugars, Iron)' +  
 'VALUES (\'' + row[27] + '\', \'' + row[8] + '\', ' + row[28] + ', ' + row[4] + ', ' +  
 row[0] + ', ' + row[10] + ', ' + row[13] + ', ' + row[11] + ', ' + row[15] + ', ' +  
 row[14] + ', ' + row[18] + ', ' + row[22] + ', ' + row[21] + ', ' + row[1] + ', ' +  
 row[23] + ', ' + row[31] + ', ' + row[29] + ')\n')  
 else:  
 x = 31  
 while x > 0:  
 if row[x] is None or row[x] == '':  
 row[x] = '0'  
 x -= 1  
 outPut.write('insert into FOOD(Name, Serving\_Info, Calorie, Carb, Fat, Protein, Fiber, ' +  
 'Cholesterol, VA, Calcium, VC, Sat\_Fat, MonoSat\_Fat, Potassium, Sodium, Sugars, Iron)' +  
 'VALUES (\'' + row[27] + '\', \'' + row[8] + '\', ' + row[28] + ', ' + row[4] + ', ' +  
 row[0] + ', ' + row[10] + ', ' + row[13] + ', ' + row[11] + ', ' + row[15] + ', ' +  
 row[14] + ', ' + row[18] + ', ' + row[22] + ', ' + row[21] + ', ' + row[1] + ', ' +  
 row[23] + ', ' + row[31] + ', ' + row[29] + ')\n')  
  
def delimit\_meals():  
 with open('E:\OneDrive\DataScience\IST 659\Project\SamsungHealth\meals.csv') as csv\_file:  
 forecast\_reader = csv.reader(csv\_file, delimiter=',')  
  
 ##initial pointers  
 count = 0  
  
 ##Full file read  
 for row in forecast\_reader:  
 if (count == 0):  
 outPut = open('E:\OneDrive\DataScience\IST 659\Project\OutPut\\' + row[0] + '.sql' , 'w+')  
 count += 1  
 continue  
 elif (count == 1):  
 count += 1  
 continue  
 elif row[12].find("'") != -1:  
 row[12] = row[12].replace("'", "''")  
 outPut.write('INSERT into MEALS(FoodID, MealDT, Amount) VALUES ' +  
 '((SELECT(FoodID) FROM FOOD where Name = \'' + row[12] + '\'), \'' +  
 row[2] + '\', ' + row[0] + ')\n')  
 else:  
 outPut.write('INSERT into MEALS(FoodID, MealDT, Amount) VALUES ' +  
 '((SELECT(FoodID) FROM FOOD where Name = \'' + row[12] + '\'), \'' +  
 row[2] + '\', ' + row[0] + ')\n')

**\*\*\* it looks a lot nice outside of a word doc.**

**Python Methods:**

I use a python module called csv to open the csv files that I export from my Samsung health and Chase and load the csv into a **for row** loop. Basically, for each row in the csv that contains useful data I pull out the columns that I decided my database would use. I ignore the data that doesn’t pertain to my project. These columns are then inserted into an output .sql file, framed in a insert statement into the a specific table.

With these python procedures I’m able to generate some insert statements for my database, below is some example inserts that were generated for each table. (Not including full inserts as the scripts are up to 5000 lines long and some have my banking info) There’s a lot more than this.

**INSERTS**

**Checking**

INSERT into CHECKING(TransactionDT, Description, Amount) VALUES ('01/29/2018', 'VENMO CASHOUT PPD ID: XXXXXXXXX',18.00)

INSERT into CHECKING(TransactionDT, Description, Amount) VALUES ('01/25/2018', 'PAYMENT FOR AMZ STORECARD XXXXXXXXX WEB ID: XXXXX',-11.99)

INSERT into CHECKING(TransactionDT, Description, Amount) VALUES ('01/25/2018', 'Payment to Chase card ending in xxxx 01/25',-293.33)

**Credit**

INSERT into CREDIT(TransactionDT, Description, CategoryID, Amount) VALUES ('12/30/2018', 'SQ \*MANHATTANVILLE COFFEE', (SELECT(CategoryID) FROM EXPENSE\_CATEGORY where Description = 'Food & Drink' ), -4.27)

INSERT into CREDIT(TransactionDT, Description, CategoryID, Amount) VALUES ('12/28/2018', 'GREYHOUND #4146', (SELECT(CategoryID) FROM EXPENSE\_CATEGORY where Description = 'Travel' ), -51.50)

INSERT into CREDIT(TransactionDT, Description, CategoryID, Amount) VALUES ('12/28/2018', 'GORDON BIERSCH SYRACUSE', (SELECT(CategoryID) FROM EXPENSE\_CATEGORY where Description = 'Food & Drink' ), -36.51)

INSERT into CREDIT(TransactionDT, Description, CategoryID, Amount) VALUES ('12/29/2018', 'PAYPAL \*STEAMGAMES', 1,-19.99)

**Food**

insert into FOOD(Name, Serving\_Info, Calorie, Carb, Fat, Protein, Fiber, Cholesterol, VA, Calcium, VC, Sat\_Fat, MonoSat\_Fat, Potassium, Sodium, Sugars, Iron)VALUES ('Banana', 'medium (7" to 7-7/8" long)', 105.0, 26.95, 0.39, 1.29, 3.1, 0.0, 2.0, 1.0, 17.0, 0.132, 0.038, 422.0, 1.0, 14.43, 2.0)

insert into FOOD(Name, Serving\_Info, Calorie, Carb, Fat, Protein, Fiber, Cholesterol, VA, Calcium, VC, Sat\_Fat, MonoSat\_Fat, Potassium, Sodium, Sugars, Iron)VALUES ('Watermelon', 'cup, diced', 46.0, 11.48, 0.23, 0.93, 0.6, 0.0, 17.0, 1.0, 20.0, 0.024, 0.056, 170.0, 2.0, 9.42, 2.0)

insert into FOOD(Name, Serving\_Info, Calorie, Carb, Fat, Protein, Fiber, Cholesterol, VA, Calcium, VC, Sat\_Fat, MonoSat\_Fat, Potassium, Sodium, Sugars, Iron)VALUES ('Strawberries', 'cup, halves', 49.0, 11.67, 0.46, 1.02, 3.0, 0.0, 0.0, 2.0, 149.0, 0.023, 0.065, 233.0, 2.0, 7.08, 4.0)

insert into FOOD(Name, Serving\_Info, Calorie, Carb, Fat, Protein, Fiber, Cholesterol, VA, Calcium, VC, Sat\_Fat, MonoSat\_Fat, Potassium, Sodium, Sugars, Iron)VALUES ('Fried Egg', 'large', 89.0, 0.43, 6.76, 6.24, 0.0, 210.0, 6.0, 3.0, 0.0, 1.88, 2.754, 67.0, 238.0, 0.38, 5.0)

\*\*\*The way my db works requires FOOD to be inserted before MEALS, the aggregate select statements in the meals insert depends on the food table)\*\*\*

**Meals**

INSERT into MEALS(FoodID, MealDT, Amount) VALUES ((SELECT(FoodID) FROM FOOD where Name = 'Coffee'), '2018-10-02 15:44:10.994', 1.0)

INSERT into MEALS(FoodID, MealDT, Amount) VALUES ((SELECT(FoodID) FROM FOOD where Name = 'Beef Stew with Potatoes and Vegetables in Gravy'), '2018-10-02 15:46:19.603', 2.0)

INSERT into MEALS(FoodID, MealDT, Amount) VALUES ((SELECT(FoodID) FROM FOOD where Name = 'Bacon'), '2018-10-02 15:46:19.603', 1.5)

**Burned Cal**

insert into CAL\_BURNED (Cal\_BurnedDT, Rest\_Cal, Active\_Cal) VALUES ('2018-11-02 04:00:01.967', 2163.3074, 11.78)

insert into CAL\_BURNED (Cal\_BurnedDT, Rest\_Cal, Active\_Cal) VALUES ('2017-10-31 04:00:00.959', 1817.2334, 117.28999)

insert into CAL\_BURNED (Cal\_BurnedDT, Rest\_Cal, Active\_Cal) VALUES ('2017-10-30 04:04:48.750', 1802.5422, 149.28)

**Steps**

insert into STEPS(StepsDT, Steps\_Count, Speed, Cal\_Burned) VALUES ('2019-07-30 11:02:44.957', 10, 2.4877067, 0.53000003)

insert into STEPS(StepsDT, Steps\_Count, Speed, Cal\_Burned) VALUES ('2019-07-30 11:07:11.443', 7, 1.3888888, 0.38)

insert into STEPS(StepsDT, Steps\_Count, Speed, Cal\_Burned) VALUES ('2019-07-30 11:07:11.454', 2, 1.3333334, 0.1)

insert into STEPS(StepsDT, Steps\_Count, Speed, Cal\_Burned) VALUES ('2019-07-30 11:16:16.363', 7, 1.5833334, 0.36)

**Pulse**

insert into PULSE (PulseDT, Pulse) VALUES ('2015-05-24 19:36:07.514', 81.0)

insert into PULSE (PulseDT, Pulse) VALUES ('2015-05-26 00:23:24.135', 76.0)

insert into PULSE (PulseDT, Pulse) VALUES ('2015-05-26 14:04:12.227', 84.0)

insert into PULSE (PulseDT, Pulse) VALUES ('2015-05-26 17:56:38.154', 80.0)

insert into PULSE (PulseDT, Pulse) VALUES ('2015-05-26 21:03:04.451', 72.0)

insert into PULSE (PulseDT, Pulse) VALUES ('2015-05-27 01:13:32.345', 74.0)

insert into PULSE (PulseDT, Pulse) VALUES ('2015-05-27 10:20:52.471', 66.0)

If you’d like full insert scripts let me know and I can email them (they’re big)

**Valuable selects:**

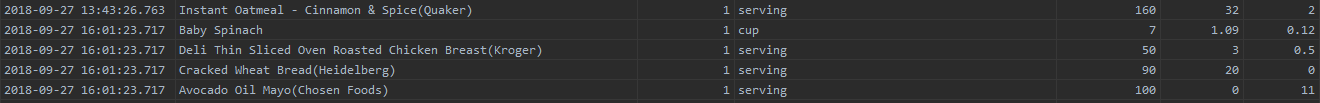
The views from my DDL are Selects that would return valuable info that I was looking for in my project 1 description. I’ll paste some results of those views now:

CREATE VIEW Meal\_Contents as

SELECT MealDT, F.Name, Amount, F.Serving\_Info, Calorie, Carb, Fat, Protein, Fiber from MEALS --view for looking at meals with their food description

JOIN FOOD F on MEALS.FoodID = F.FoodID

go;



CREATE VIEW Credit\_History as

SELECT TransactionDT, CREDIT.Description as Description, EC.Description as Category, Amount --view for looking at credit history joined with categorys

from CREDIT

join EXPENSE\_CATEGORY EC on CREDIT.CategoryID = EC.CategoryID

go;



CREATE VIEW Credit\_Balance as

SELECT ROUND(SUM(Amount), 2) as Balance

FROM CREDIT

go;

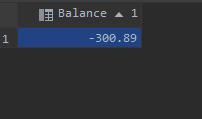


CREATE VIEW Checking\_Balance as

SELECT ROUND(SUM(Amount), 2) AS Balance

FROM CHECKING

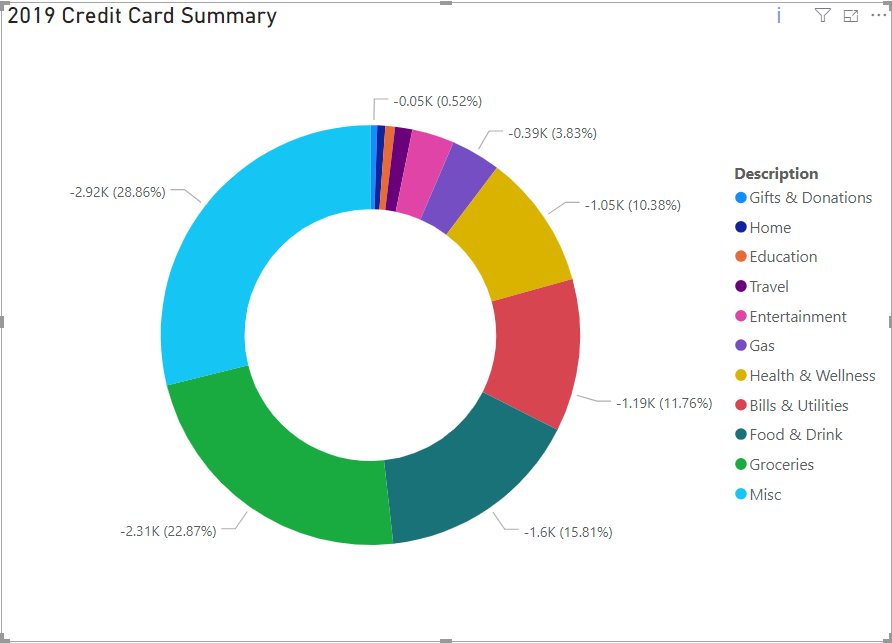
go;



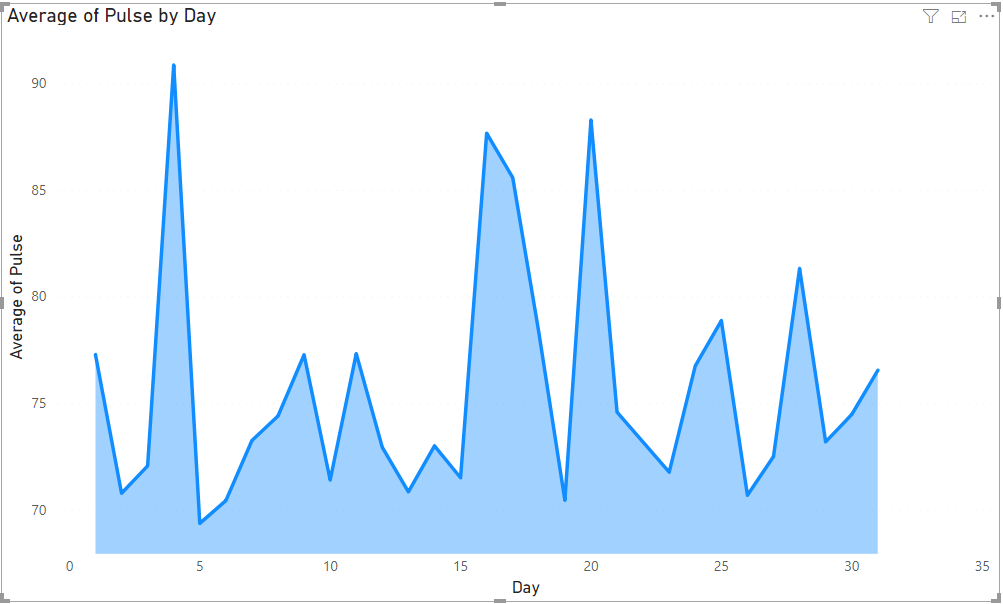
The latter two of these views are not 100% accurate as the state of my balances before these exports where taken was not 0 dollars so things are a little skewed in that regard. But the views work as expected.

**POWER BI / FRONT END**

I tied my database directly into Power BI. As a first-time user of Power BI I found it pretty easy to create some very significant graphs of my data that more than satisfy my original project pitch. Using power BI also saved a lot of time that I would have spent using python to graph my data.

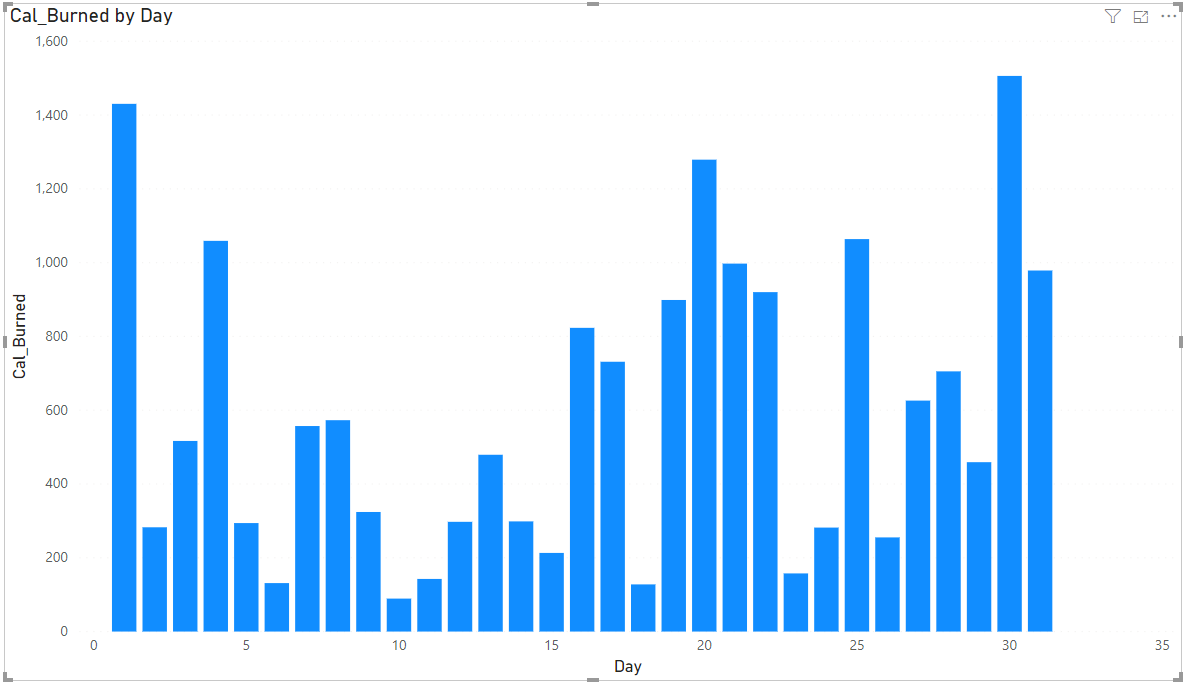
First up is a graph of my Credit spending joined to their respective categories:

Second is a line graph of my average pulse for the last week:

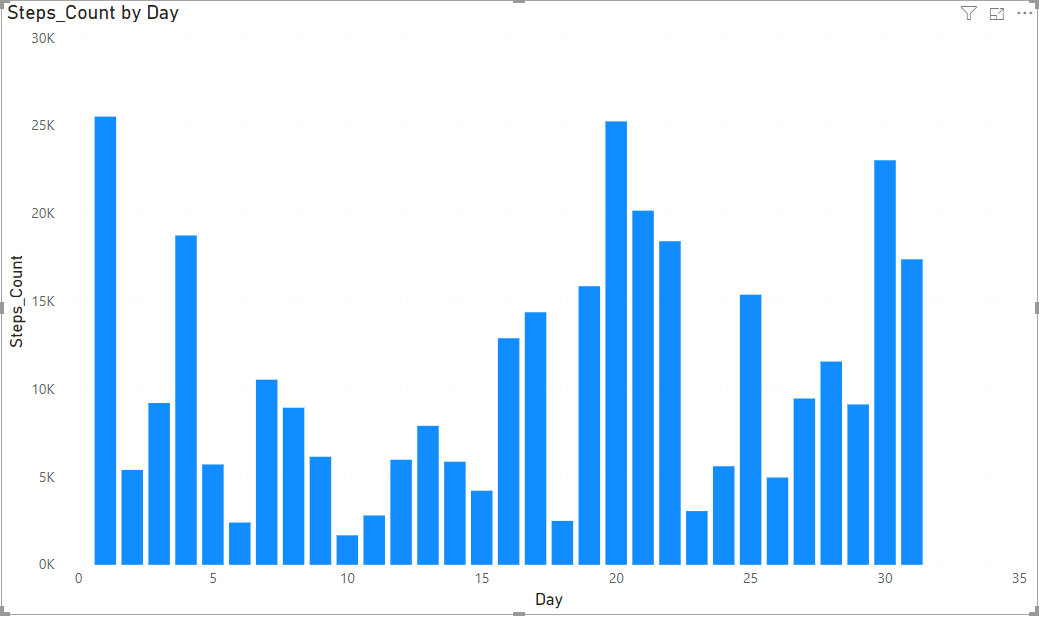


From this graph you can see spikes on different days, these are days that I’m going on runs. Turns out a 5K can really jack up my average heart rate for the day, even if its only an hour long run.

This graph correlates closely to my next graph Cal Burned **by Steps** per day.



Further more backed up by graphing Steps Count by day, pretty much a 1:1 correlation with Cal\_Burned by day:



**Reflection**

When I started this project I had some pretty lofty ideas, I even stated that in my initial summary and methods section, and it really proved true in my implementation. Also my assumption that delimiting my google data would be easy was not true at all, the way those exports are structured makes it far more complex to delimit and I just gave up.

I think this project was a great refresher for my knowledge of sql, I rarely deal with table creates or procedures in my regular job and revisiting them really expanded my understanding of sql. I also jump at any opportunity to sharpen my python skills, and my insert design was quite fun to do. My decision to use Power BI was a easy one as well, I was dreading scripting the modeling in python, but after our lab about Access and class discussions about Tableu and Power BI I decided to give the latter a try, and wow was that easy. I was able to pick up the application quickly and activated my license via my syr.edu email. I really like Power BI and it gives me some open possibilities for creating some more complex graphs of my data.

I’ll be implementing some of what I learned into my regular job, scripting SQL statements via python will be very useful, and now I have a much deeper understanding of what our DBA’s do. And I feel much sharper in my knowledge of VIEWS, PROCEDURES, and FUNCTIONS. All in all, I do thing this project was enjoyable and a great exercise for my working knowledge of SQL and Database Management. I hope to continue to use the scripts and database I’ve built to further track my goals and life as a whole.