Emilija Dimikj

June 10, 2020

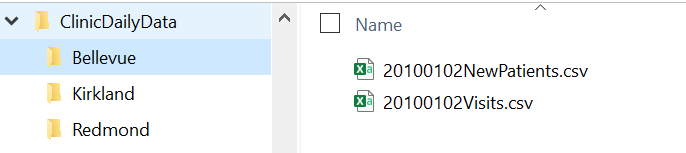
BIDD 220 B Sp 20: Data Migration Techniques (ETL Processing)

Final Project

Final Project - Data Migration Techniques (ETL Processing)

# Milestone 01: File Imports

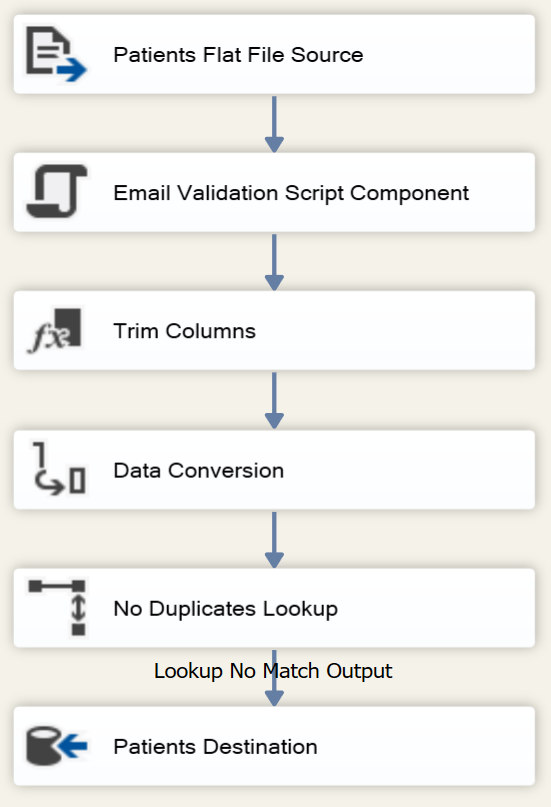
In this milestone we are making ETL process of loading CSV files to local SQL Server existing database. CSV Files are uploaded to the file directory daily and each clinic has its own folder where the data files are uploaded *(Figure 1.1)*.



*Figure 1.1: File directory structure and content*

|  |  |
| --- | --- |
| ETL process is done in SSIS package  **M1-OLTPSourceFileCleaning** *(Figure 1.2)*  Patients and Visits ETL processes are independent processes but for easier debugging and following the process they are executed in succeeding order.  All errors and warnings are logged into files.  Both Foreach Loops are taking the main folder *...ClinicDailyData* as input folder and then traverse the subfolders. Looped file name is passed in a variable for the usage in the Data Flow Task. | *Figure 1.2: Milestone 1 SSIS package* |

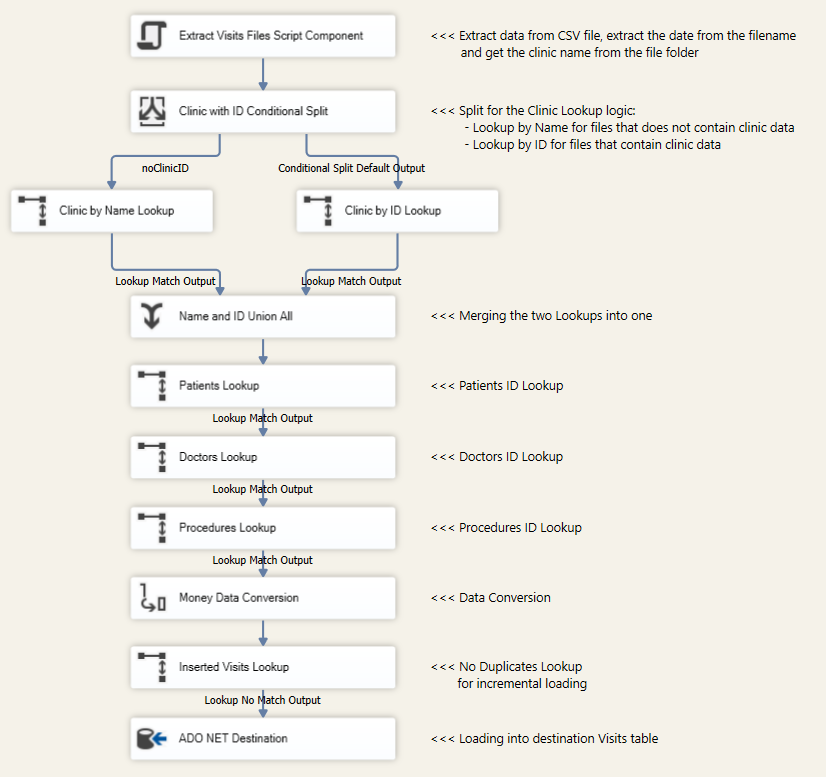
Patients Data Flow Task *(Figure 1.3)* reads the Patients CSV file via Flat File Source, do validation of the email values via Script Component using C# code. Invalid emails are put in a log file together with a filename and date and time of validation. In the next two steps columns are transformed and before loading the data into destination Patients table No Duplicates Lookup is performed for the purpose of Incremental Load. In this case we are just inserting new patients and not receiving any updated or patients that needs to be deleted.



*Figure 1.3: Patients Data Flow*

Visits Data Flow *(Figure 1.4)* imports data from CSV file. Since that file contains just new visits data, we are not implementing any updates or delete of existing rows in destination table.

Date extraction, Clinic Name extraction for the files that does not contain clinic information and merging date from the file name and time from the data in the files is done via C# script.



*Figure 1.4: Visits Data Flow*

# Milestone 02: Data Warehouse ETL

This ETL process is between the **DoctorsSchedules** and **Patients** OLTP databases and the **DWClinicReportDataEDimikj** OLAP database. The two OLTP databases are on different servers, one hosted on Azure and the other on the UW network.

**Patient** OLTP Database and **DWClinicReportDataEDimikj** database are on UW SQL Server**: is-root01.ischool.uw.edu\BI**

**DoctorScedules** OLTP database is on **continuumsql.westus2.cloudapp.azure.com**

In the time of development of the ETL process one of the source servers and destination server were not linked and that was the reason for using stage tables. All the source data from **DoctorScedules** OLTP database is first dumped into staging tables from SSIS without any transformations and all transformations are done using ETL views on staging tables. This approach appeared to be only possible way of getting the data and latter on proved as a good and fast for Transformation and Loading stage. Data from **Patient** OLTP Database is transformed using ETL views directly on source tables since this and destination database are on the same server.

SSIS Package M2-ETL for DWClinicReportData contains ETL dataflow and all code is in SQL scripts. SQL code is encapsulated in stored procedures. ETL process is done in 4 steps:

1. Check Remote Server*(Figure 2.1)*:

|  |  |
| --- | --- |
| First Check DW Database SQL Task run check to see if DataWarehouse already exists on the remote server. This task fills the *User::varDatabaseCount* variable with 0 if DW doesn’t exist or with 1 if DW exist.  If DW is not created yet, runs Create DW SQL Task (5-Create DWClinicReportData.sql) and when database is created executes Create DW ETL Views (6-CreateViews DW.sql) that create all views used in the ETL process and last executes Create DW Fill Procedures (7-Create LoadingProcedures DW.sql) and create all stored procedures. Check DW Database Again SQL Task is identical as the firs task in this step and is run to reset the value of *User::varDatabaseCount* variable so that process can continue on the next step.  If DW exists go directly to the next PreLoad ETL step.  No mater how mush times this package will be executed, this step is executed just once. | *Figure 2.1: Check Remote Server* |

1. PreLoad ETL:

|  |  |
| --- | --- |
| PreLoad ETL stage *(Figure 2.2)*, is preforming flash and fill for staging tables. It truncates all staging tables and populate them with data. Loading data is done with SSIS Data Flow Task *(Figure 2.3)*, extracting the data with ADO.NET Source and loading to ADO.NET Destination without any transformations.    *Figure 2.3: Load to Stage Table* | *Figure 2.2: PreLoad ETL* |

1. Fill Dim Tables ETL:

|  |  |
| --- | --- |
| This step is populating all Dimension Tables with data and is executed after successful execution of the PreLoad ETL process *(Figure 2.4)*.  First Insert Dummy Rows SQL Task calls the script saves in *8-DimTab Dummy Rows.sql*. The script first check if dummy row already exists and if not inserts one in each Dim table with ID=0 and resetting Identity\_Insert *(Figure 2.5)*. In case of NULL values in any of Fact Tables this row will be matched during transformation stage.  Each next step is executing store procedure for populating Dimension Table with data accordingly. **DimDates** store procedure generates dates in date range. **DimPatients** procedure synchronize source and destination data using MERGE statement and Type 2 SCD. **DimClinics**, **DimDoctors**, **DimShifts** and **DimProcedures** procedures synchronize source and destination data using MERGE statement and Type 1 SCD. | *Figure 2.4: Fil Dim Tables ETL* |

*Figure 2.5: Code for inserting dummy row*

Declare @hasDummyRow int;

select @hasDummyRow=count(\*) from [dbo].[DimDoctors] where [DoctorID]=0;

if @hasDummyRow=0

begin

--insert Dummy Row for NULL handling

Set Identity\_Insert [dbo].[DimDoctors] On;

Insert Into [dbo].[DimDoctors](

[DoctorKey], [DoctorID], [DoctorFullName],

[DoctorEmailAddress], [DoctorCity], [DoctorState], [DoctorZip])

Values(

0, 0, 'Unknown Doctor',

'Unknown', 'Unknown', 'Unknown', '00000')

Set Identity\_Insert [dbo].[DimDoctors] Off;

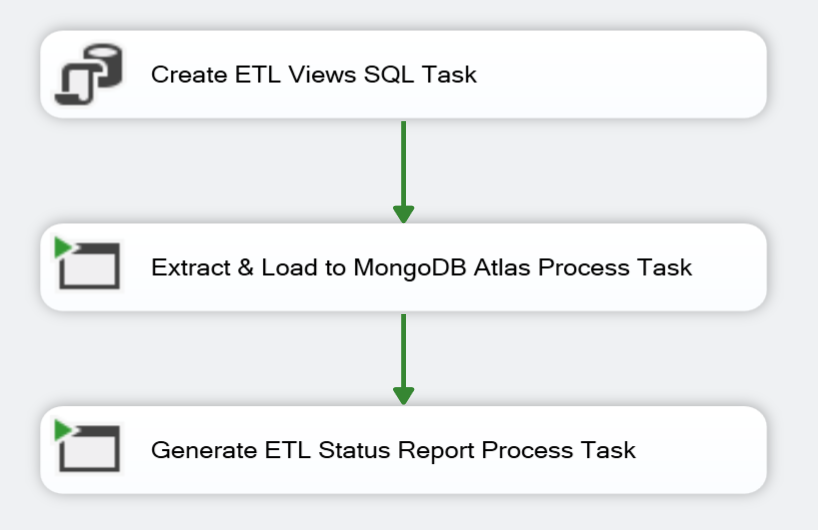
end

1. Fill Fact Tables ETL:

|  |  |
| --- | --- |
| This step is populating all Fact Tables with data and is executed after successful execution of filling all Dimension Tables with data *(Figure 2.6)*.  FactVisits Execute SQL Task calls stored procedure dbo. pETLFillFactVisits and populate the **FactVisits** table with data.  FactDoctorShifts Execute SQL Task calls stored procedure dbo. pETLFillFactDoctorShifts that populate the **FactDoctorShifts** with data. | *Figure 2.6: Fill Fact Tables ETL* |

# Milestone 03: New Feature Evaluation

In this Milestone, we are creating ETL process for transferring data from **DWClinicReportDataEDimikj** database hosted on is-root01.ischool.uw.edu\BI SQL Server to MongoDB database hosted on Atlas. Python proved to be one of the best and fastest ways to work with MongoDB. Whole ETL process is done in Python and Python Script is executed from SSIS package *(Figure 3.1)*.



*Figure 3.1: ETL For MongoDB SSIS Package*

SSIS package is executed in three steps:

1. Creating the ETL Views in Data Warehouse

Data for extract from Data Warehouse is organized in two views, one for each collection:

* vETLVisitsCollection: contains data for loading to VisitsCollection
* vETLDrShiftsCollecton: contains data for loading to DrShiftsCollection

Views script file is saved as ..\SQLScripts\CreateCollViews.sql and executed from SSIS package.

1. Extract and Load Data to MongoDB Atlas



*Figure 3.2: Python function for returning the connection string for SQL Server*

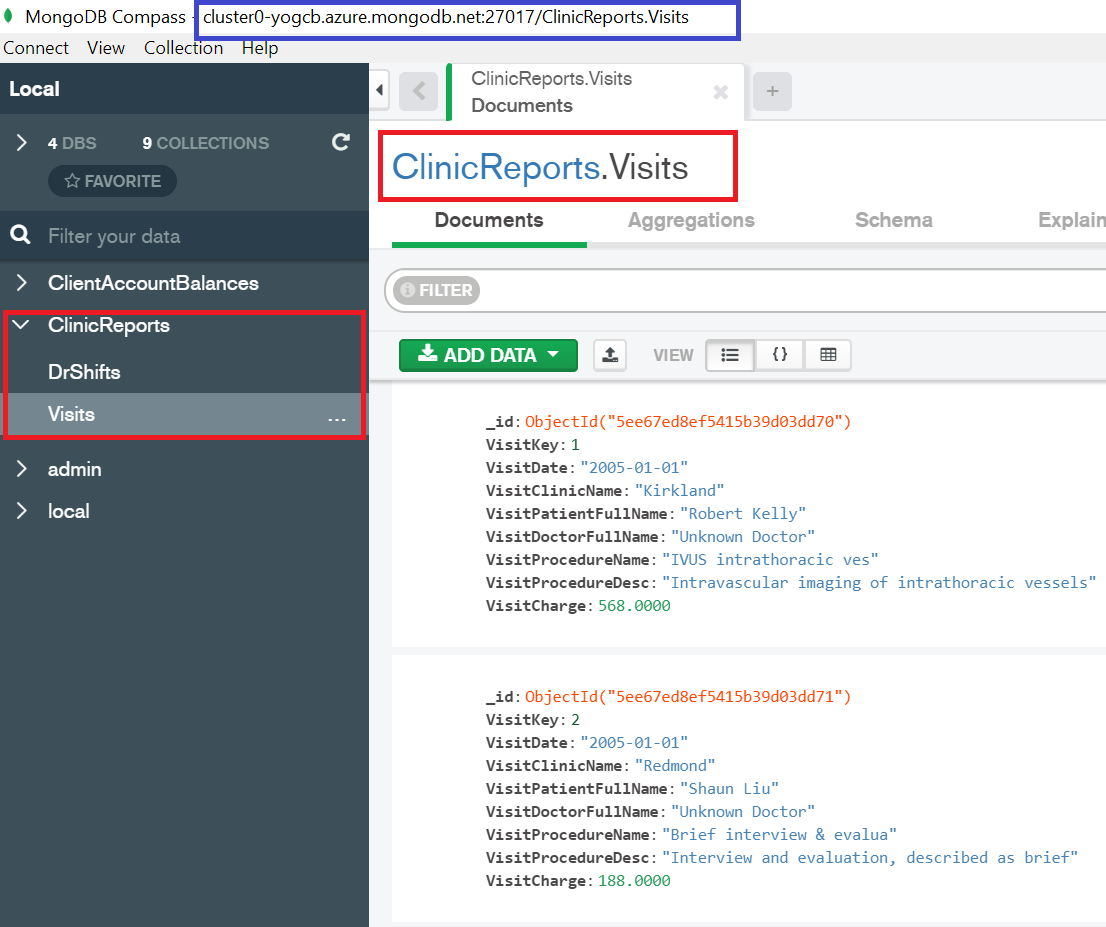
Python Code is organized in functions.

* getDWConnectonString() function *(Figure 3.2)* returns SQL source server’s connection string
* getMongoConnectionString() function returns MongoDB Atlas source connection string
* VisitsColl() function extract data from vETLVisitsCollection view on remote SQL Server and transform data ready for load
* DrShiftColl() function extract data from vETLDrShiftsCollecton view on remote SQL Server and transform data ready for load
* Load2Mongo() function creates mongo database, drops collection if exist and creates new and load it with data
* fillClinicReportDatabase() function *(Figure 3.3)* is calling data extraction and loading functions.



*Figure 3.3: Python function for organizing ETL process flow*

After executing package **ClinicReportsDatabase** with two collections: **VisitsCollecton** and DrShiftsCollection is created and populated with data *(Figure 3.4).*

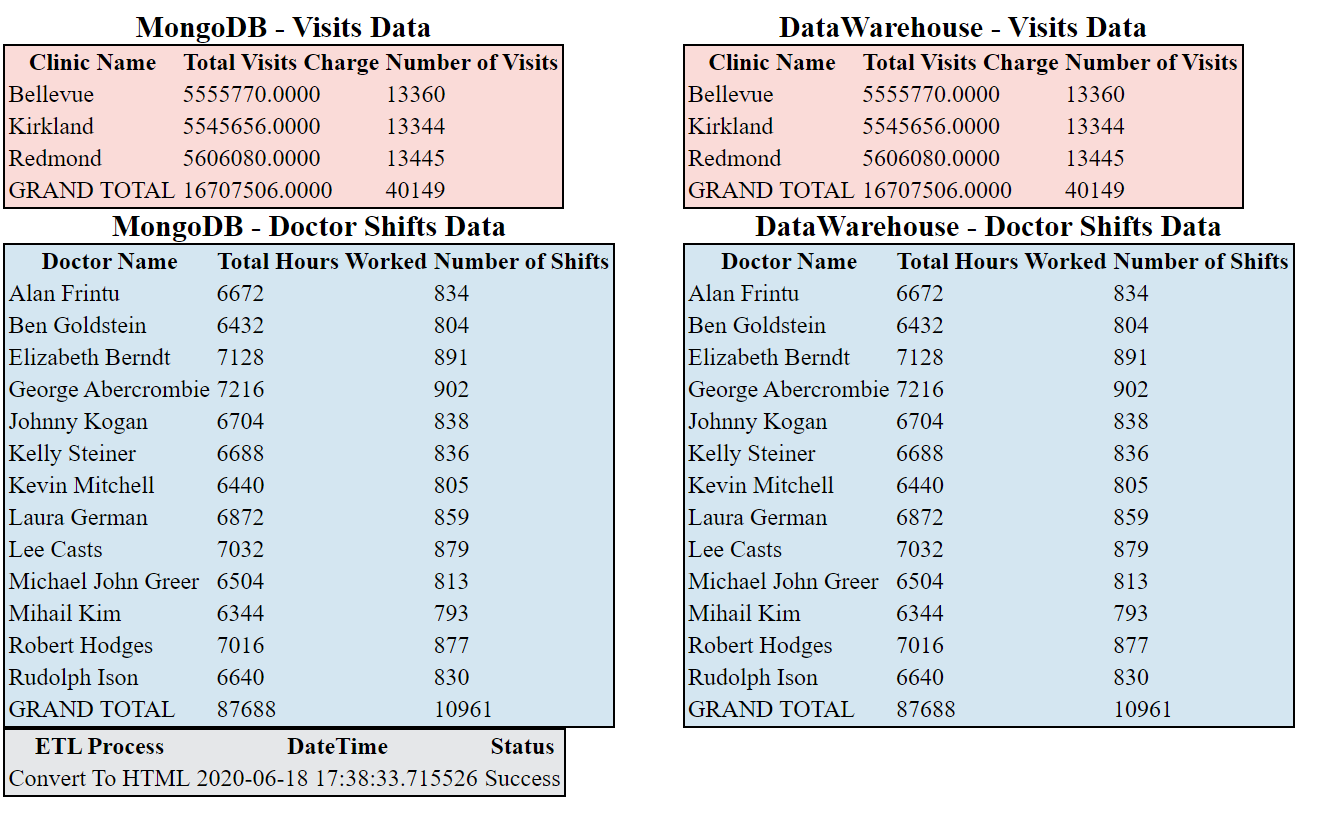


*Figure 3.4: Database and Collections created on MongoDB Atlas*

Making extraction, transformation and loading data all in python didn’t required cleaning special characters and commas from columns that contain text. With this, original text is preserved. Having commas in the text means that we need to take that into consideration in case of eventual export to SCV or text file with comma delimiter.

1. Generate ETL Status Report

Last step is to create comparation data report for the ETL process, to check if there is any difference between source and destination data *(Figure 3.5)*



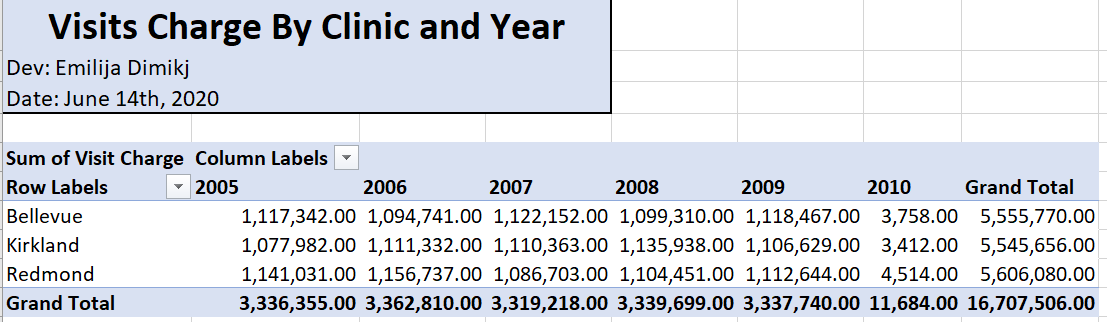
*Figure 3.5: ETL report*

# Milestone 04: Reports and Documentation

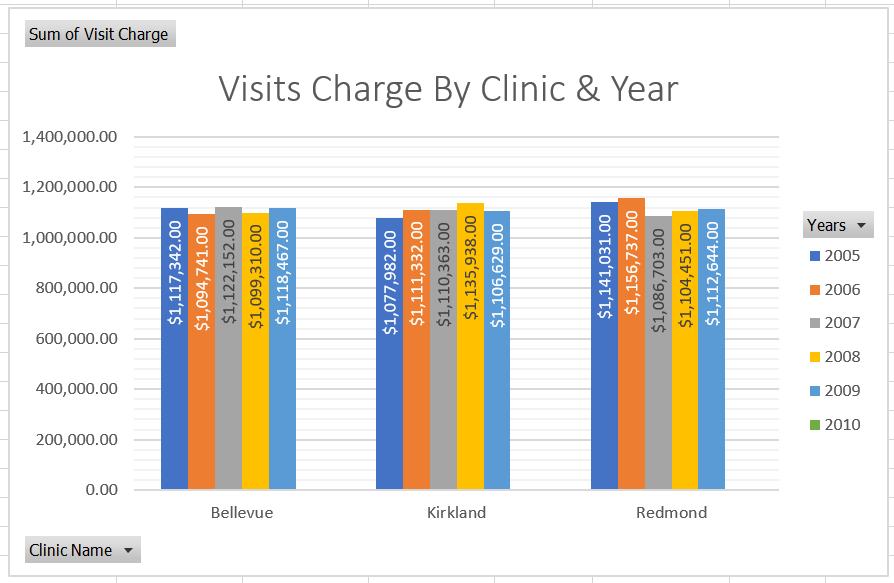
Last milestone of this final project is to create few Excel reports from SQL Server’s Data Warehouse and MongoDB Atlas.

1. Data Warehouse Report

Connection between Excel and **DWClinicReportDataEDimikj** is done using integrated data importer from SQL Server. Raw data is transformed and imported into DWVisitsReportData tab. That raw data is used as a source of pivot table to create ***Visits Charge by Clinic and Year*** report *(Figure 4.1)* accompanied with graph for better visual presentation *(Figure 4.2)*.



*Figure 4.1: Data Warehouse Report’s pivot table*

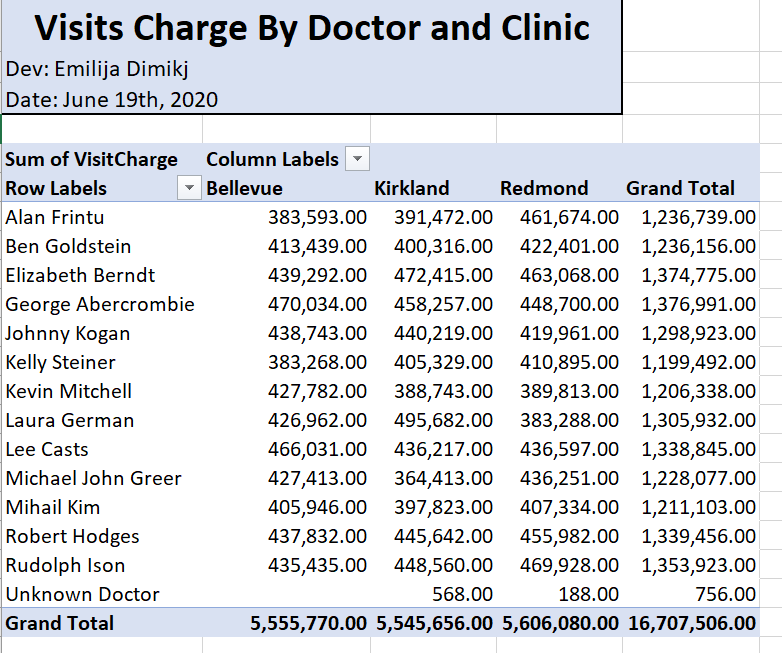


*Figure 4.2: Data Warehouse Report’s graph presentation*

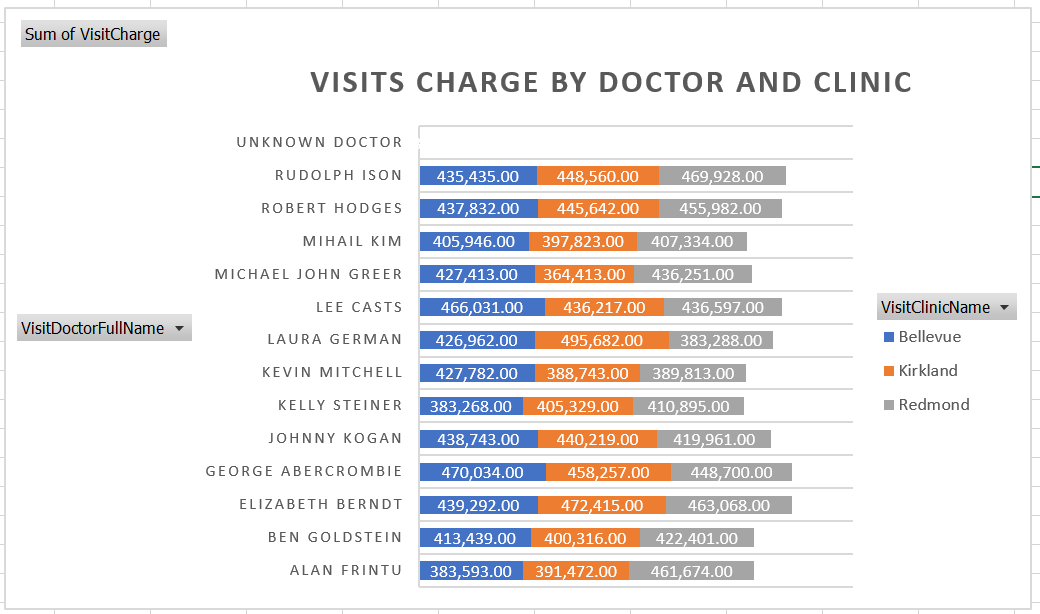
1. MongoDB Report

In this project my MongoDB Atlas is on M0 Sandbox Cluster Tier and is free MongoDB service. MongoDB has ODBC driver for connection to the Atlas but for this tier level is not working, it requires to be M10 and up. Without working driver I was unable to connect Excel with MongoDB Atlas.

To get report data I created Python script for extracting data and saving it to CSV file. Python script is executed from SSIS package **M4-ExtractReportData**. The script is saved under name and it uses **pymongo** and **csv** libraries to extract report data. That CSV file is transformed and loaded into report Excel file into MongoDBVisitsReportData tab. That data is used as a source of pivot table to create ***Visits Charge by Doctor and Clinic*** report *(Figure 4.3)* accompanied with graph for better visual presentation *(Figure 4.4).*



*Figure 4.3: MongoDB Report’s pivot table*



*Figure 4.4: MongoDB Report’s graph presentation*

# Summary

In this project I tried to implement all knowledge gained during this course. First milestone is fully done in SSIS and C# script, second in SQL and third in Python utilizing power od SSIS for the dataflow. At the same time learned new things, used different approaches than before and explored python libraries, unknown to me.

Writing this technical document was a recap of all work done on this project. Retracing the steps and retesting the packages and code, I encountered few problems and found few things that I could have done differently. There is always space for improvement.