The main scope of this project is to deploy an automated data collection solution using the Watson Studio data science platform in conjunction with a IBM DB2 on Cloud database. In developing this solution, my intention was to solve a problem that was mentioned by one of the instructors from the Data Science Methodology course of this program. During one of the course presentations, the instructor had placed an emphasis on mentioning that one of the most difficult and time-consuming aspects of the data science process is data collection, processing, and generation of new data. While I'm not a data scientist by any means, I can relate to and agree that the amount of worked involved in preparing data for any kind of analysis and, or research is not only difficult, but exhausting. This is the type of work that often has little to no reportable production output and depending on the circumstances, can come under a great deal of criticism as the allocation of human capital towards such efforts are costly.

By using a series of multiple of Jupyter notebooks and a cloud database, I was able to not only automate the data collection process in retrieving data from the Foursquare API database, but also generate new data as well. When a user performs a location-based trending search using any of the Foursquare applications or API calls, the search results will only return trending data based on the number of users currently checked in at a location at the time the search is executed. Once the browser or application is closed, the user no longer has access to the information provided in that search. Which is one of the issues this project solves, by simply executing an API call that performs a trending search and stores the search results in a database for permanent storage. In doing this, the database will have historical time-series trending venue data capable of producing useable data in real-time with preestablished time delay.

Now, I'm sure Four squares already have this data and offers it to customers through one of its premium subscription services, which starts off at around $600 a month, but I couldn’t find solid historical data of this nature through any other channels offered with the unpaid developer’s subscriptions. The closest thing I found was the Popular response field under the Get Venue Details endpoint API call. This endpoint will give users a range of operating hours when the specific location searched has the most amount of traffic. This information doesn't give me a weighted value of any nature to compare how popular the venue is at that time in relationship to other venues.

I'll be fourth coming in pointing out that intent of this project was to develop a solution which aims to accurate the data collection and processing stages of the data science methodology. Most of my time was focused on developing a solution to improve aid in future data science projects to come. I was able to generate new time series data using the trending venue API call. This data can be viewed in the Analysis and Report notebook stored in my GitHub repository, which is one of the 15 notebooks utilized for this project. The following list provided below contains links to the notebooks that we're utilized throughout the course of the project, not including the numerous testing files generated throughout the troubleshooting process.

Web Scraper Notebook

Data Retrieval Notebook

Analysis & Report Notebook

The data contained in the Analysis and Report notebook is a combination of data generated by the Web Scraper notebook that was used in the Data Collectors, along with new data produced by the data collectors. For the actual data that was used or, can be used to perform an analysis, I was able to collect trending venue data for various cities in five-minute intervals spanning a period of 5 hours and 36 minutes on Sunday, Jan. 20th, 2019. While this is not a significant amount of data, it was generated autonomously using the following structure shown in the diagram below.

The Data Retrieval notebook was created as a work around and will eventually be eliminated, unless another purpose for it is found. This was added to the scope of the project as a temporary solution for exceeding my maximum daily API call limits which is capped at 950 calls per day. One of the problems I've been having throughout the course of the project, is inconsistent data being returned when performing an API call to the Foursquare database. Based on my observations, the search results vary based on the city and venue identified in the search. While I'm not completely sure, I believe venue plays a more significant role in the variations, however the differences tend to be somewhat consistent from one city to the next. In addition to this, I've noticed that since I've began working on this project, the incoming data has been changing leading me to be believe that the records contained in the database are being updated simultaneously. The reason this becomes an issue, is I've been working to program exceptions into the data collectors in order to produce consistent data frame priors to submitting the data frame to the DataSciDB database.

If you review the results generated by the data collectors in the Analysis & Report Notebook you'll notice that the data isn't uniform in the dataframe. In order to produce an auto generated data sample for this project, I had to let start letting some of the unprocessed data through to the final dataframe in order to get results sent to the DataSciDB database for storage. From there the data is retreived and processed in the Data Retrieval notebook.

The most critical aspect of the project is the data collectors, which handle all of the hard work by performing API calls, retreiving the search results, processing the incoming data, and sending it to the database for retreival. Originally, I only intended to use four data collectors that would be scheduled to run every hour 15 minutes apart. As a means of producing a large enough data set for finishing this project, I increased that number to 12 and scheduled each one to run hourly, five minutes apart from one another. This resulted in a lot of overlapping data that probably isn't necessary, but it does provide a good example of the system capabilities available with the IBM Watson Studio platform. Once I started letting unprocessed data through, the program worked excellent. I monitored the Trending Data table in the DataSciDB during the process and the database was automaically refreshing as new data came in.

Going forward, if I can continue to add exceptions to accommodate the search result variations, the Analysis & Reporting can be scheduled to run hourly, so every time a user we're to access the notebook, they would be looking at updated data results and visualizations without having to rerun the program. This program still needs a lot of work, with a focus on creating a secondary database table to act as a cache for results falling outside of current exception parameters. This will be helpful in planning future adjustments and prevent program errors that are likely to occur when data is sent to the database.