



8/8/2023

Cloud Miners

Analyzing Music Market Data for
Artists and Listeners

Lani Lewis, Haitie Liu, Lijo Jacob, Mai Dang & Todd Garner

Term Project
SMU DS 7330 Summer 2023

Table of Contents

Group Name:.....	2
Introduction:	2
Project Overview:.....	2
Steps Taken:	3
Action Items:	5
Conclusion:.....	6
Appendix A SQL ERD:.....	7
Appendix B URLs:.....	7
Appendix C Process:	8
Appendix D Data Processing:.....	10
Appendix E Azure Rhythm Architecture:	13
Appendix F Machine Learning:	19

Group Name: Cloud Miners

Introduction:

This proposal outlines the project plan for the "Cloud Miners" group, composed of members Lani Lewis, Mai Dang, Haitie Liu, Lijo Jacob, and Todd Garner. The objective of this project is to gather listening data for specific artists and their tracks, based on various locations. The collected data will be stored in an Azure database created by the team. The primary goal is to capture information to enable comprehensive analysis of a music market for a given artist. To present the findings in an accessible manner, the team plans to develop a visual representation of listening activity, which will be made available to the public through an R Shiny app (Azure Rhythm). Below, you will find a broad overview of the process undertaken with individual sections with more detail about each activity found in the Appendices.

Project Overview:

The Cloud Miners team aims to gather and analyze listening data for artists and their tracks. By identifying music market information, the team intends to gain insights into the distribution and popularity of certain artists in different regions. The following steps outline the project workflow:

1. **Data Collection:** The team will utilize various data sources, such as streaming platforms and music services, to gather listening data for specific artists and their tracks. The data will include information on the music market by artist.
2. **Azure Database Creation:** A dedicated Azure database will be set up by the team to store the collected data securely. This database will serve as the central repository for all the information gathered during the project. We will identify the relationships between multiple entities and use the data to retrieve metrics for the visual representation that will be used in the R Shiny App.
3. **Analysis:** The team will leverage the music market metrics available from the collected data to perform detailed analysis of artist distribution. This analysis will help identify trends around high listener activity and potential areas for targeted marketing or performance planning.
4. **Visual Representation:** Using the analyzed data, the team will develop a visual representation of the music market activity. The visuals will also include statistical information around market trends for a particular artist. The visuals will be interactive, allowing users to explore and interact with the data.
5. **R Shiny App Development:** To make the project findings accessible to the public, the team plans to create an R Shiny app. This app will host the visual representation of the music market trends and metrics, allowing users to explore and query the data based on their preferences. It will provide an intuitive and user-friendly interface for interacting with the collected information.

Steps Taken:

We used 'Spotify' library in Python, a well-structured interface that allowed us to query and obtain large data sets from Spotify. This data set, enriched with diverse elements such as artist names, album titles, popularity, followers, loudness, instrumentalness, acousticness, and more primarily subject categories and the number of streams, was then recorded into a CSV file for efficient data manipulation and analysis. In conjunction with the Spotify data, we also obtained music information from other sources like Kaggle.com and Billboard.com and their affiliate.

Subsequently, these CSV files were uploaded into an Azure Rhythms database. The Azure platform offered us the robustness to handle large-scale data and provide the necessary tools to sort and analyze effectively.

1. Initial Planning and Proposal Preparation:

- Reviewed the project's objectives and outlined the proposal.
- Identified the need to create a database (DB) to hold Music information.
- Planned to use the Spotify R Package to extract data in Python from Spotify and store it in the DB.
- Aimed to develop an R Shiny App or R Markdown that could access the DB and provide users with music statistics.

2. Research and Data Analysis:

- Explored available datasets from Kaggle and other websites.
- Explored Java/JSON and R-based globe visualizations for integration into the project.
- Explored data sources for storing music information.
- Chose R Shiny as the visualization tool for the project.

3. Finalizing Project Proposal:

- Drafted the project proposal, considering the research findings.
- Defined the group and project names.
- Studied a proposal sample shared on GitHub for reference.
- Prepared a summary

4. Database Design and Architecture:

- Developed an Entity-Relationship (ER) diagram to establish relationships between datasets.

- Designed an end-to-end architecture for the project.

5. Azure Integration and Data Loading:

- Explored working with Azure or R for data-related tasks.
- Utilized Azure Data Factory to pick up data from CSV files and load it into the Azure database.
- Focused on retrieving market information, Is local status, and market country codes for artists and tracks.
- Explored DVI library, SQL query views, and the Azure Rhythm Dashboard for additional functionalities.
- Upload CSV files to the SQL database for storage.
- Leveraged Azure's Query Editor to view and analyze the tables.

6. Team Collaboration and Azure Setup:

- Collaborated to ensure all team members are able to connect to Azure database.
- Set up Azure Data Studio for querying the data.
- Continued gathering information about songs and artists from sources like Billboard and other relevant sources.
- Focused on expanding knowledge of Azure and Azure Data Studio as a Team

7. Database Optimization and Data Loading Completion:

- Updated the Entity-Relationship Diagram (ERD) (Appendix A).
- Initiated table creation (Appendix A) and documented queries and steps.
- Received guidance from Lijo on proper indexing techniques.
- Loaded the remaining datasets into the Azure database.

Our Cloud Miners team successfully designed and implemented a pipeline to extract, manage, and analyze vast amounts of music streaming data. This exploration will serve as the basis for our innovative application that will combine analytics and visualization to help a stakeholder make informed projection decisions.

Action Items:

Our intentions go beyond just analyzing the data set; we aim to develop an R ShinyApp that would provide an interactive visualization of our findings.

One of the standout features of this application would be a global representation indicating the number of streams from various geographical locations. This would offer insights into the popularity and reach of the artist's music. We believe this application would be an invaluable tool, empowering artists with real-time insights from Spotify.

1. Data Retrieval:

- Determined meaningful queries for data analysis.
- Perform numerous information retrieval methods utilizing Python through API's (application programming interface) with various sources
- Pull data and clean it to eliminate duplicates

2. Analytical Decision:

- Determined which Statistical test/s will be used
- Ensure queries will provide the data needed for R Shiny App

3. Machine Learning to derive our objective:

- Pull a table of metrics from our project database
- Load this table into Azure Machine Learning Service and Azure Machine Learning Studio
- Process pertinent information from Azure ML Studio and other tools used
- Produce a usable equation with an estimated formula for Popularity
- Test it against existing Popularity values for fit

4. R Shiny App

- Complete the R Shiny App (Appendix B)
- Utilize the results from the Machine Learning exercise above
- Build numerous useful applications within R Shiny.io for users to review
- Test the App

5. Final Draft and Presentation

- Create Final Draft
- Determine presentation flow
- Practice presentation before due date
- Deliver presentation on due date




Conclusion:

The Cloud Miners project aimed to gather, clean, normalize, remove duplicates, and analyze music market data for selected artists (approximately 2,195 artists) and their tracks (approximately 7,687 tracks) across various metrics. By utilizing an Azure database and performing statistical analysis, the team intends to provide valuable insights into artist distribution and a predictive formula for estimating “Popularity.” The visual representation of music market trends, integrated into an R Shiny app, will make the findings readily available to the public. This project holds significant potential for music industry professionals, artists, and marketers, empowering them to make informed decisions based on music market engagement.

Each section above is explained with accompanying screenshots in the following Appendices.

Appendix A | SQL ERD

This is the ERD Diagram for the Azure Rhythm Database. As well as the SQL queries to set up the table schemas for the database.

Linked File	Description
 ERD_Diagram.pdf Click the icon to the left to see the ERD Diagram.	This is the pdf file of the AZURE ERD Diagram
 SpotifyERD_Version 1.mwb Click the icon to the left to view the MySQL workbench ERD used in this project.	ERD MWD file for use with My SQL
 CreateTable.sql Click the icon to the left to see the SQL queries used to create tables in this project.	Create Table SQL Queries

Appendix B | URLs – R Shiny Apps (click the links below)

URL	Site Description
https://azrhythmwebapp.azurewebsites.net/	Azure Rhythm Home Page
https://haitieliu.shinyapps.io/FileclassRshiny1/	Azure Rhythm R Shiny App

Data Preparation and Import

Setting up the environment by importing necessary libraries for data manipulation, visualization, web application creation, and more. It also reads in multiple CSV files, creates a temporary image file, and sets up a hyperlink to a GitHub repository.

Dashboard Layout Definition

The dashboard is made up of three key components: a sidebar, a header, and a body. The sidebar includes various user inputs (sliders, dropdowns, checkboxes, and date range selectors) that are contextually presented based on the active tab.

The header includes the title of the dashboard, linked to a GitHub repository.

The body consists of multiple tab panels, each dedicated to a specific type of analysis (Word Cloud, Time Series, Song Features, Popularity Prediction, and Globe) and corresponding to different visualizations which depend on user inputs from the sidebar.

The output is rendered based on user choices using R's plotly, shiny, and other related packages. The 'dashboardPage' function is then used to combine these components and the final dashboard user interface is stored in 'ui'.

Server Function and Shiny App Execution

This is the server function for the Shiny app, where each output object (i.e., plots, text, images) is created based on the user's inputs and rendered for the dashboard. The 'renderText', 'renderUI', 'renderImage', 'renderPlot', and 'renderPlotly' functions are used to create the outputs, which include calculated results, plots, and images.

The popularity score is calculated from a pre-determined formula, while the other outputs are dynamic, generated through various types of plots and tables. The plots include word clouds, time series, bar charts, boxplots, and parallel coordinate plots, each corresponding to different tab panels from the UI.

The plots are constructed using a combination of base R functions, ggplot, and Plotly. The 'globe' output renders a 3D globe representation of streaming data. Preprocessed data is loaded and manipulated based on the inputs received from the user, and the results are visualized in the appropriate format.

Finally, the 'shinyApp' function is used to run the application, linking the defined user interface ('ui') and server-side operations ('server').

For more details: <https://haitieliu.shinyapps.io/RshinyAppMarkDown/>






Appendix C | Processes

After we as a group finished analyzing the raw data and created relational tables. We were next tasked with finding queries to use for retrieving useful data for our Machine Learning and ultimately R Shiny App.

At first the idea was to create a materializing view so that we would have a snapshot of the information and the query would not take too long to run. We attempted to create a view and ran into problems with Azure and copyrights issues. Due to time constraints, we decided the next best approach was to create procedures which we could use in the R Code to retrieve the data from the server.

The first procedure created was an input procedure or a procedure that would accept input values. The procedure was called "Globe" and has two input variables: "2017/2018", and "Artist/Track". As a team contributed queries and plotted ideas that we thought would be good for our R Shiny App. The remaining procedures do not have input variables.

Click the icon in the right-hand column to see each query used in our processes.

Procedure Name	Description	SQL Query
globe	Allowing the users to visually see top ranks by Artist or Track and by Year 2017 or 2018. This is for the Globe Visual.	 globe.sql
song_duration	Compare the average duration of songs over the years.	 song_duration.sql
loudness_over_time	Compare the average loudness of songs over the years.	 loudness_over_time.sql
popularity_vs_followers	If you have a lot of followers does that make you more popular?	 popularity_vs_followers.sql
trackFeatures	Do certain features improve a track ranking?	 trackFeatures.sql

Next, we have included the R Markdown HTML output which was used to test that the procedures were working correctly and would allow us to pull the data in the R Shiny App. We also conducted a small EDA to confirm that the statistical concept would work with the data we have and queries we moved forward with.

Click the icon below to see numerous queries and their output



SQL_QUERIES.html

Appendix D | Data Processing

Extracting data using an existing Spotify API and a Python package called Spotipy to acquire and transform The Data.

This Step primarily involved:

- Utilize the data from two Kaggle datasets we acquired and extract a substantial amount of metrics from Spotify using the Spotify API, Python and the Spotipy library.

Goals:

- Obtain as much information as possible.
- Clean and process the data to ensure the accurate querying.
- Use the Spotify API to extract additional information regarding the artists and their individual tracks.
- Gain insights into any limitations of the Spotify API and how to work around them
- Learn to use the Spotipy library for interacting with the Spotify API in Python.

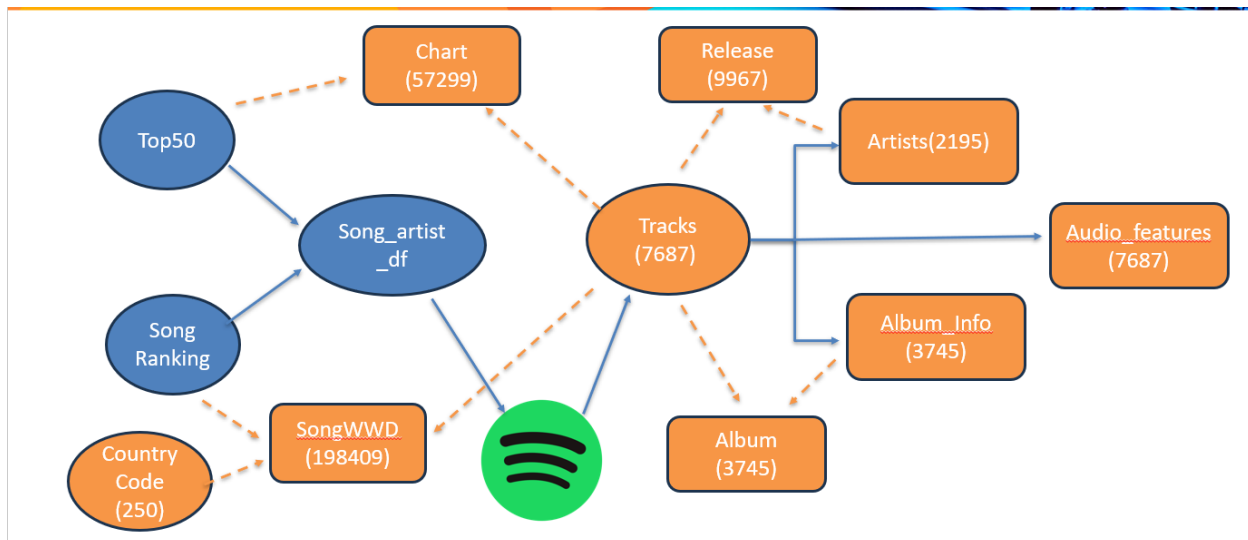
Data sources:

- Kaggle - <https://www.kaggle.com/datasets/edumucelli/spotify-worldwide-daily-song-ranking>
- Spotify.com

Data Pre-processing

The first step was to clean and preprocess the datasets to ensure consistent and accurate querying of the Spotify data via their API. The cleaning process involved:

- Handling missing values: Check for missing values in track and artist names
- Removing duplicates
- Formatting names: standardizing track and artist names to remove special characters, spaces, or any other inconsistencies.
- Normalizing the artist names



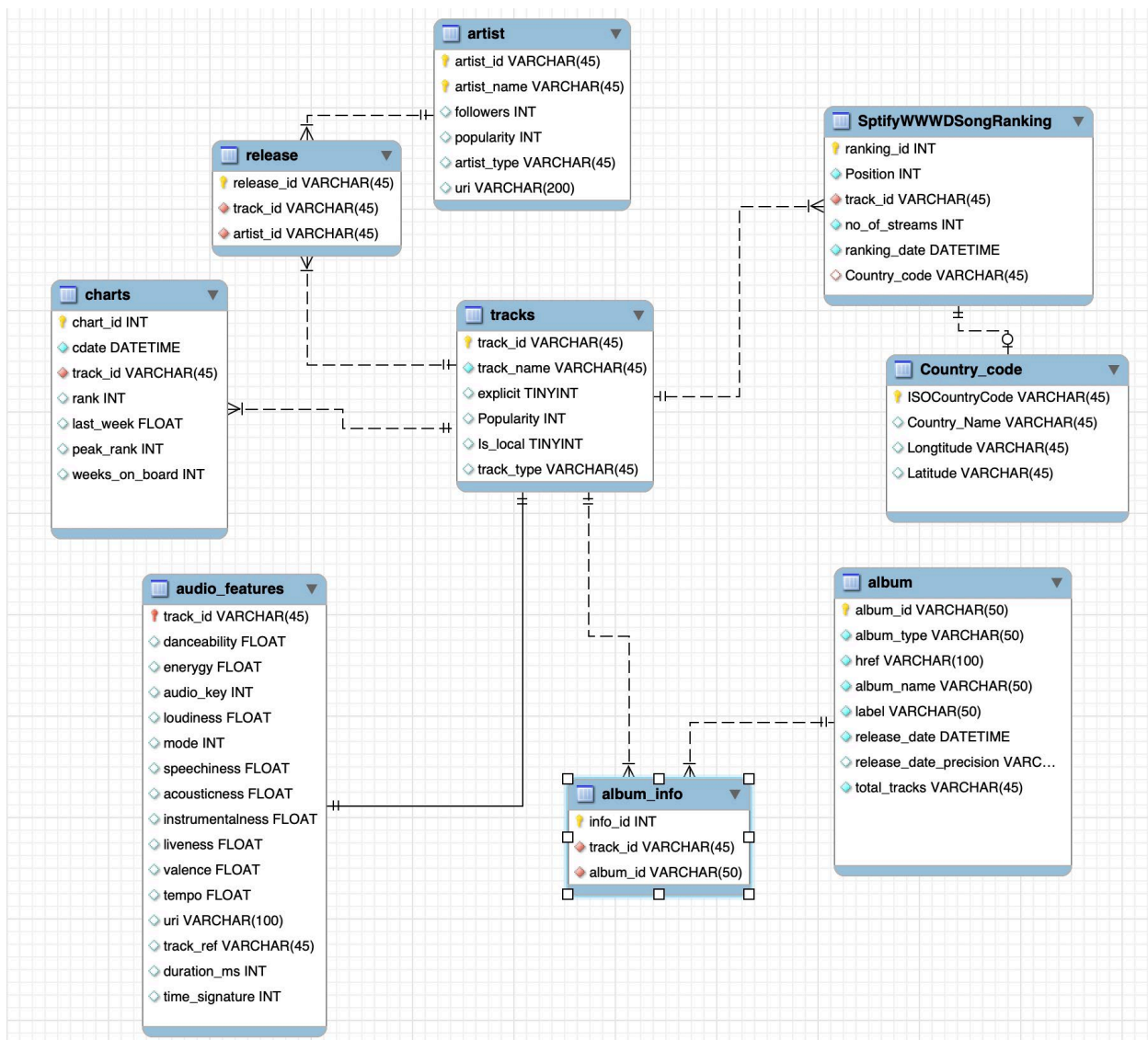
Interacting with Spotify API

Once the datasets were cleaned, the Python - Spotipy library was utilized to interact with the Spotify API. Spotipy is a Python library that provides easy access to the Spotify Web API.

- Authentication: before making any API requests, authentication was set up to access the Spotify API securely. This involved obtaining a client ID and client secret from the Spotify Developer Dashboard and using them to authenticate access to the application.
- Querying the API: For each track and artist name combination, an API request was made to Spotify to retrieve additional information about the track and artist. The data retrieved included track popularity, release date, audio features (e.g., danceability, energy, valence), and artist details.

Insights into Spotify API Limitation

- Rate limitation: Spotify API enforces rate limits on the number of requests that can be made in a given time frame. To avoid hitting these limits, rate-limiting strategies were implemented, such as adding delays between consecutive requests.
- Data availability: Not all tracks and artists returned complete information. Some data points were missing or not accessible due to privacy settings or licensing restrictions.
- Authentication process: Setting up the authentication process and obtaining the client ID and client secret initially caused some confusion, but it was eventually resolved.
- Token limitation: Each token authenticated is only valid within an hour. The issue was resolved by creating multiple client IDs and client secrets as well as adding a delay once the token limit was met.



Key Learnings

- Data cleaning and preprocessing: The importance of cleaning and preprocessing data before utilizing APIs to ensure accurate and efficient querying.
- Authentication for API access: Understanding how to authenticate applications to access restricted APIs securely.
- Spotipy library usage: Gaining hands-on experience with the Spotipy library and its functionalities for interacting with the Spotify API.
- Working around Spotify API limitations: Learning to handle rate limiting and dealing with missing or restricted data in the API responses.

Appendix E | Azure Rhythm Architecture

Azure Rhythm Architecture

Spotify API and Kaggle data sets were used as the data sources for the entire project. Python processes were created for the extraction of data from the Spotify API for various aspects for music data such as track, album, release, etc. The Python processes were enhanced to cleanse both Kaggle datasets and data from Spotify API. CSV files are created for each type of data using the Python processes at the end of this step.

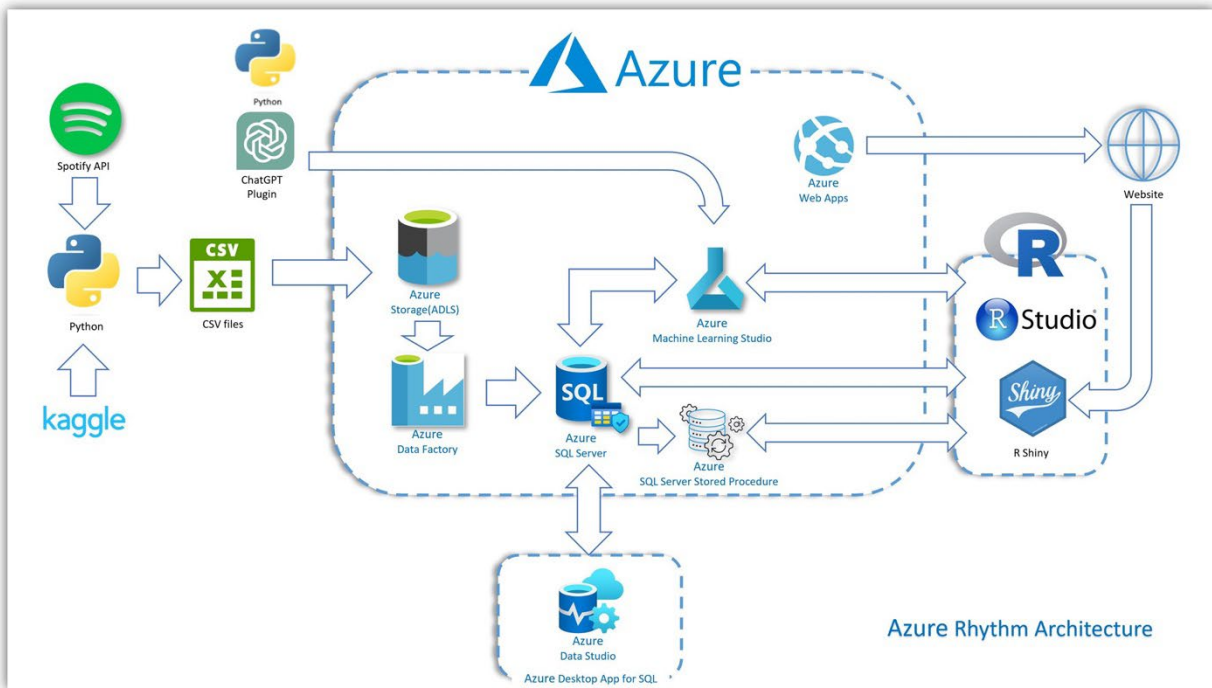
We set up an Azure account for each member of the team and created an Azure Data Lake storage account. We then migrated the *.csv files acquired in the data acquisition process into Azure Data Lake storage account. Data in *.csv files are ingested into Azure SQL server database using Azure Data Factory. These tables were then used to create relational database tables based on the ER Diagram.

Azure data studio and R Studio were used to access Azure SQL Server database from our laptops. Azure SQL Stored Procedures were created for various interactive visualization requirements in the R-Shiny App.

Azure Machine learning studio is used along with ChatGPT plugin for prediction of the Popularity predictor.

RStudio was used to develop R Shiny app to create various interactive visualization, predictions, and plots.

Azure webapps were used to create and host websites.



Azure Data Migration and Database Setup

Azure Account Setup

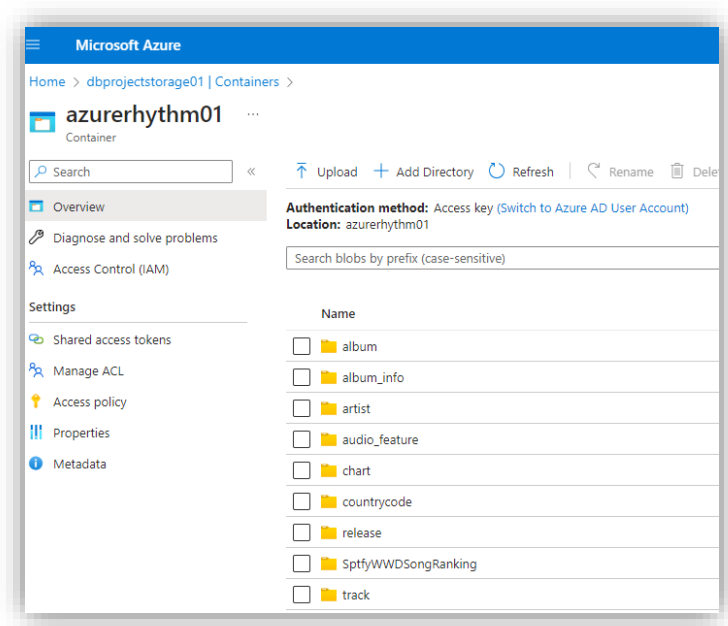
Azure student accounts were set up for each team member. Under one of our Azure Subscriptions, an Azure Resource Group was created for the project. All the resources required for the project were created under this Resource Group. Each team member is assigned a “Contributor” role.

File Migration to Azure Data Lake Storage Account



We performed the steps below as part of *.csv file migration to Azure Data Lake Storage Account

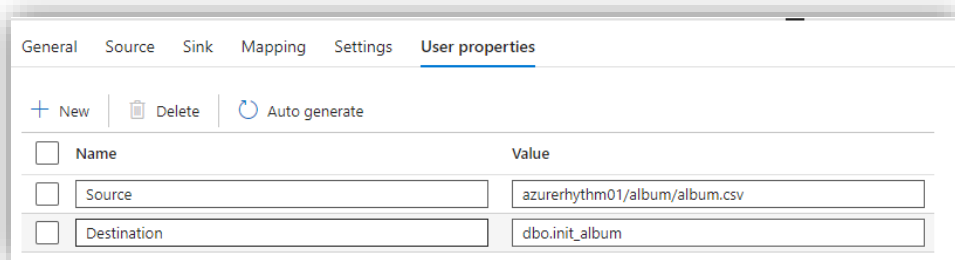
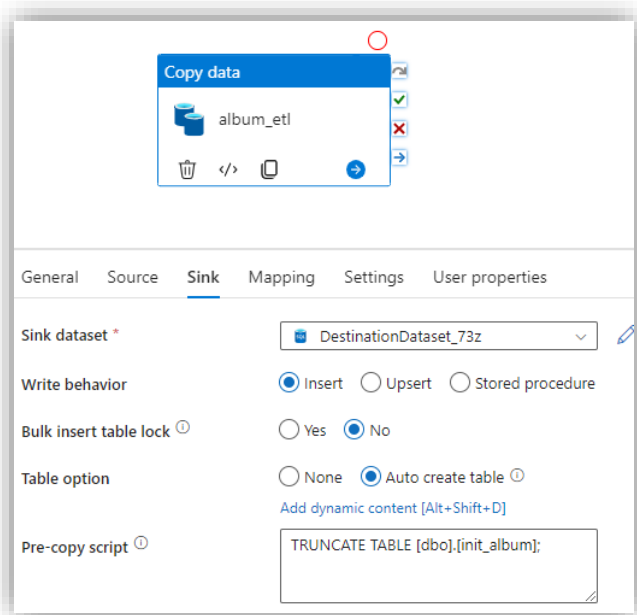
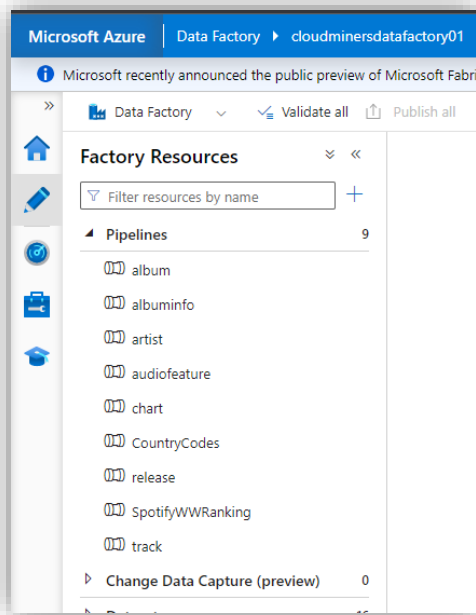
1. Setup an Azure Data Lake Storage Account.
2. Created a Container for the project
3. Created directories under this container for each file
4. Data stored in csv files was migrated to the corresponding directories in Azure data lake storage.



Data Ingestion to Azure SQL using Azure Data Factory



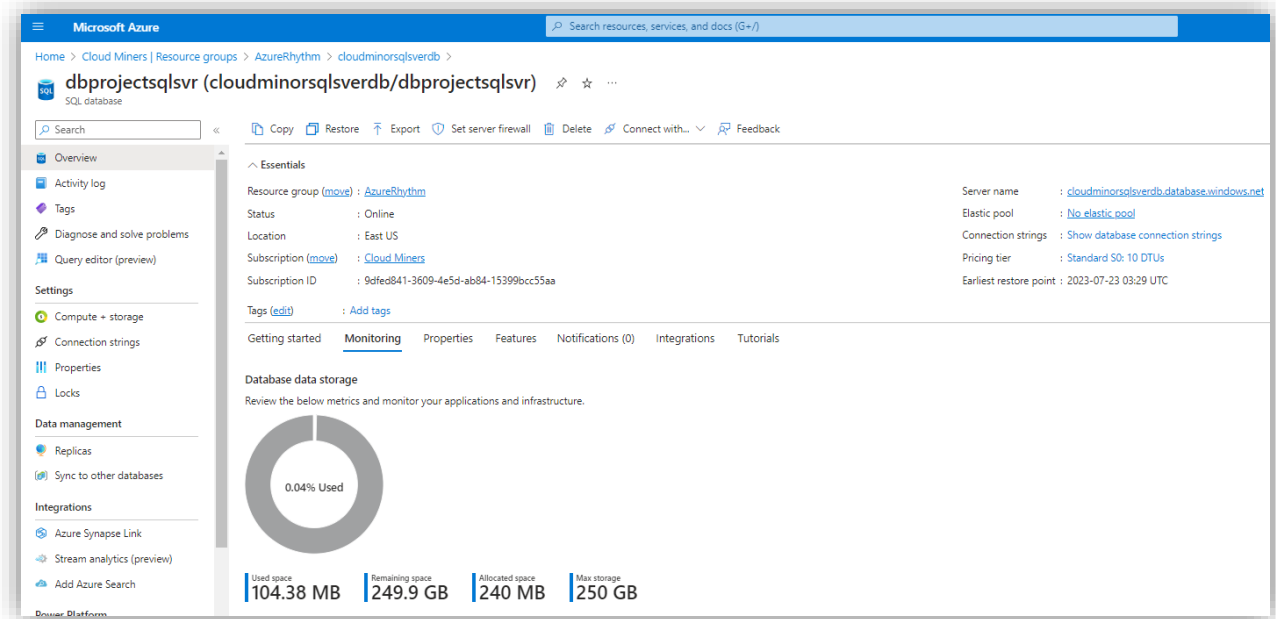
Data stored in Azure Data Lake is ingested into Azure SQL using Azure Data Factory. Data Pipelines were created for each file. Each pipeline had a data copy activity with the source being the Azure Data Lake and target being the Azure SQL Server. Each copy activity has a pre-copy script to truncate the target table before loading data. Table name format is `init_<csv_file_name>`, as shown below.



Azure SQL Database Setup



SQL server was set up first and a SQL database with 250 GB of storage with 10 DTUs (database transaction unit) as processing speed was created under the SQL Server primary account, as shown below.



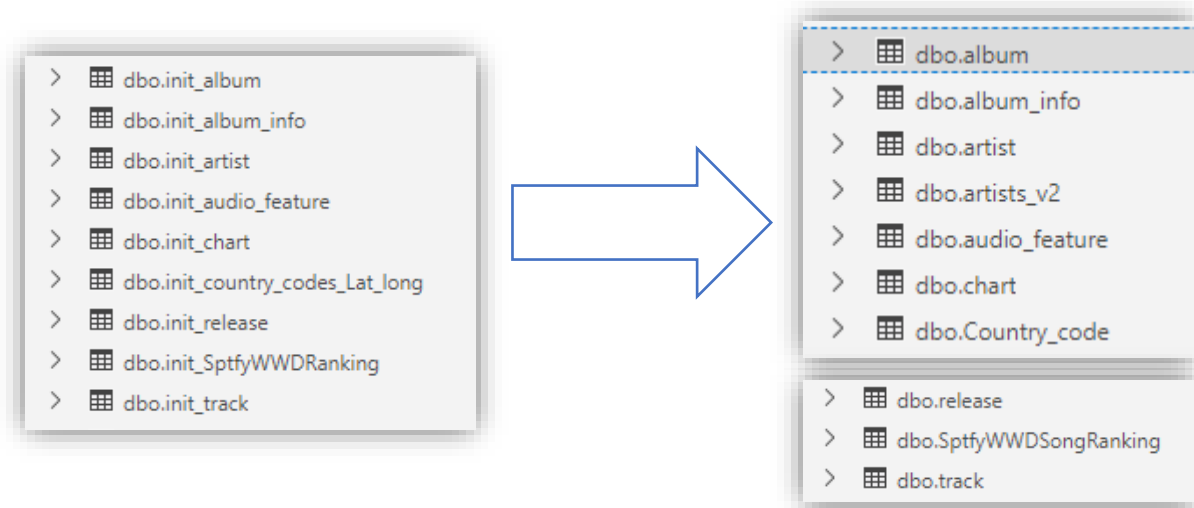
Firewall rules were set up to allow team member's local machines to access the Azure SQL server database using Azure Data Studio and the statistical software package R.

The screenshot shows the 'Firewall rules' configuration page in the Azure portal. It includes a header with the title 'Firewall rules' and a link to 'Learn more'. Below the header, there are two buttons: '+ Add your client IPv4 address (203.113.113.113)' and '+ Add a firewall rule'. The main content area displays a table of existing firewall rules:

Rule name	Start IPv4 address	End IPv4 address	
ClientIPAddress_2023-6-4_12-12-55	203.113.113.113	203.113.113.113	
ClientIPAddress_2023-6-4_18-48-35	203.113.113.113	203.113.113.113	
ClientIPAddress_2023-6-4_18-48-37	203.113.113.113	203.113.113.113	
ClientIPAddress_2023-6-4_18-48-38	203.113.113.113	203.113.113.113	
ClientIPAddress_2023-6-4_18-49-57	203.113.113.113	203.113.113.113	

Relational Database Tables Setup

For data ingested using Azure Data Factory, the landing tables are named as init_<csv_file_name>. Relational tables are created with key constraints and appropriate data types as per the ER Diagram.



Each table was created with Primary Key and/or foreign key/s using create table statement (see example below). Data was inserted into the table from the corresponding landing table with appropriate foreign key table lookup.

```
CREATE TABLE [dbo].[audio_feature](
    [af_track_id] [nvarchar](50) PRIMARY KEY NOT NULL,
    [Danceability] [float] NOT NULL,
    [Energy] [float] NOT NULL,
    [Audio_key] [tinyint] NOT NULL,
    [Loudness] [float] NOT NULL,
    [Mode] [tinyint] NOT NULL,
    [Speechiness] [float] NOT NULL,
    [Acousticness] [float] NOT NULL,
    [Instrumentalness] [float] NOT NULL,
    [Liveness] [float] NOT NULL,
    [Valence] [float] NOT NULL,
    [Tempo] [float] NOT NULL,
    [Uri] [nvarchar](150) NOT NULL,
    [Track_herf] [nvarchar](150) NOT NULL,
    [Duration_ms] [int] NOT NULL,
    [Time_signature] [tinyint] NOT NULL,
    FOREIGN KEY (af_track_id) REFERENCES [dbo].[track] (track_id)
);

insert into dbo.audio_feature select * from dbo.init_audio_feature where af_track_id in (select track_id from dbo.track);
```

Take Away/Lessons Learned

Working on Azure was a great learning experience. Especially setting up the resources and setting up permission for all team members. To have a variety of components and integrating them to make it work together seamlessly was such a great learning experience.

Phase 2. Action Items/how we could improve the utilized processes

Instead of using *.csv files, Azure Storage Account and Azure Data Factory, if we could connect from python directly to Azure SQL Server and load data, that would have been an added advantage.

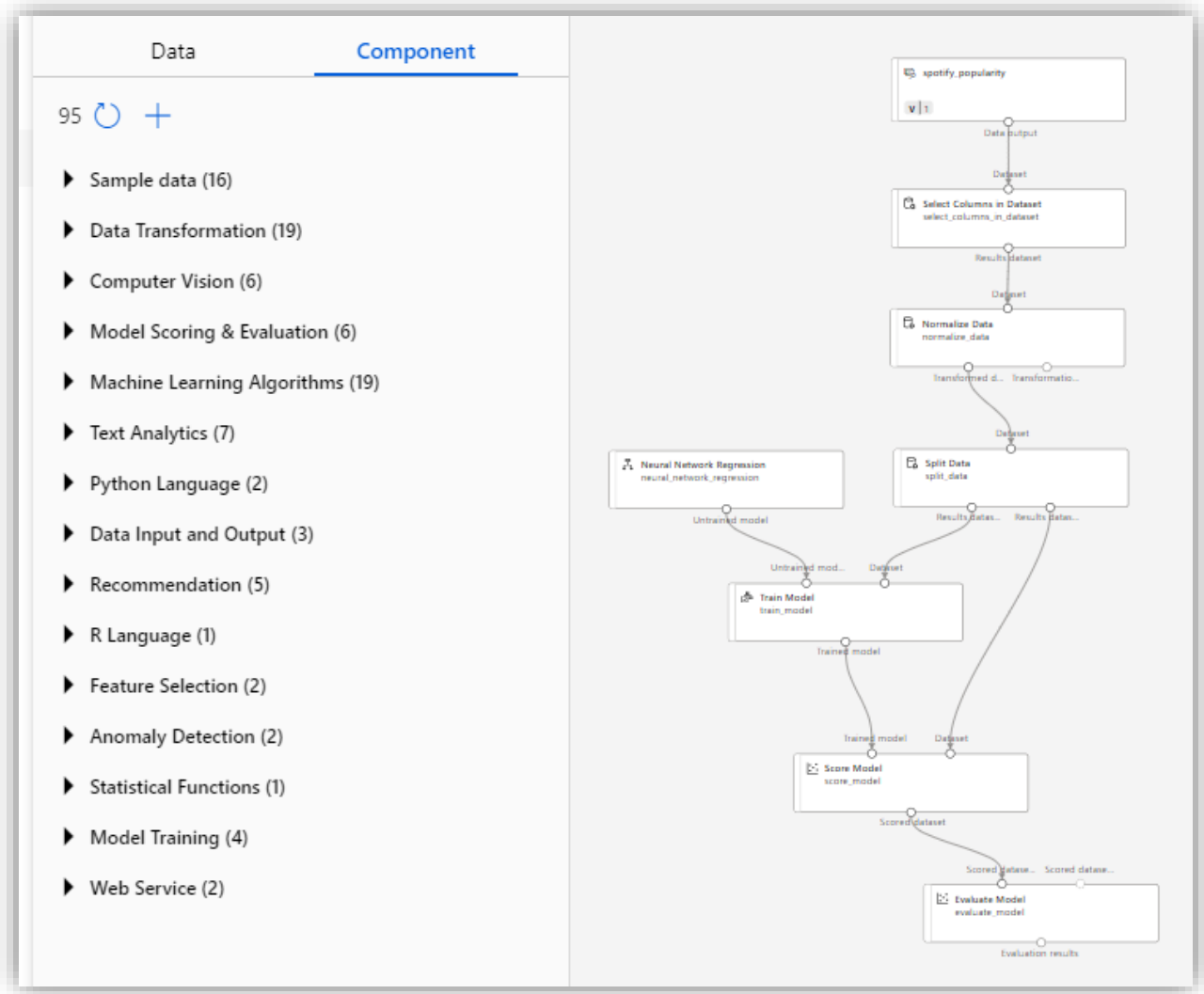
Appendix F | Machine Learning

Machine Learning in Azure – Machine Learning Studio

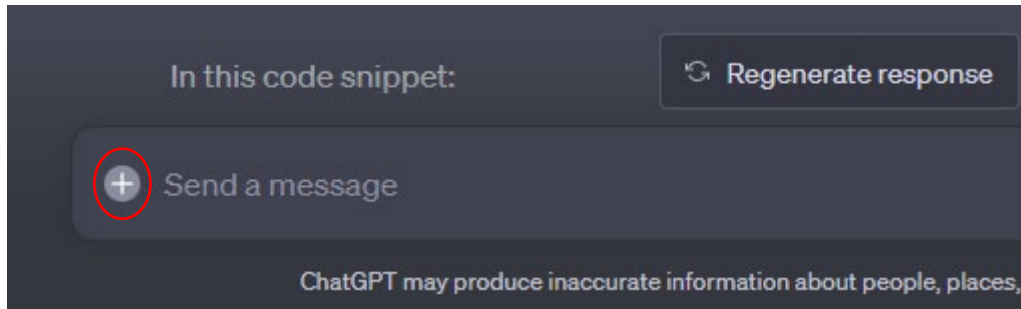
After successfully establishing databases and their tables in Azure, we initiated the application of Azure Machine Learning Services to explore the possibility of constructing a linear regression equation with as many as 11 independent variables and a single response variable - popularity. In other words, we wanted to see what it took to create a popular song. The variables were extracted from our data from Spotify and Kaggle and stored in our Azure SQL Server database. These included columns with data on popularity, followers, danceability, energy, loudness, mode, speechiness, acousticness, instrumentalness, liveness, valence, and tempo. Granted, many of these are subjective or nebulous but perhaps they've got a more common meaning within the music industry.

popularity	followers	Danceability	Energy	Loudness	Mode	Speechiness	Acousticness	Instrumentalness	Liveness	Valence	Tempo
100	78485332	0.519	0.527	-7.673	1	0.0274	0.075	0	0.132	0.267	78.915
100	78485332	0.354	0.267	-13.69	1	0.0281	0.731	0.000402	0.0858	0.113	94.219
100	78485332	0.602	0.736	-5.778	1	0.0338	0.00196	4.57E-05	0.105	0.471	96.969
100	78485332	0.624	0.757	-2.94	1	0.0296	0.00265	1.87E-06	0.189	0.658	121.07
100	78485332	0.777	0.357	-6.942	1	0.0522	0.757	7.28E-06	0.108	0.172	139.883
100	78485332	0.472	0.701	-3.72	1	0.0279	0.091	0	0.23	0.304	147.854
100	78485332	0.392	0.574	-9.195	1	0.17	0.833	0.00179	0.145	0.529	81.112
100	78485332	0.636	0.402	-7.855	1	0.031	0.0494	0	0.107	0.208	125.952
100	78485332	0.58	0.491	-6.462	1	0.0251	0.575	0	0.121	0.425	76.009

Next, to gain a comprehensive understanding of the application of machine learning in Azure, a Coursera course on "Machine Learning using Azure Machine Learning Services and Machine Learning Studio" was taken. After careful step-by-step analysis of the methods available in the Azure Machine Learning Studio and optimizing several sample queries as well as our data, the Azure help files revealed that a direct method to obtain the desired linear equation (response = $a + bx$) was not currently available within the Azure platform. (However, I do recommend both services (Coursera & Azure machine learning) highly as they are enormously powerful means to learn a fantastic machine learning engine inside the Azure ecosystem, including AutoML)



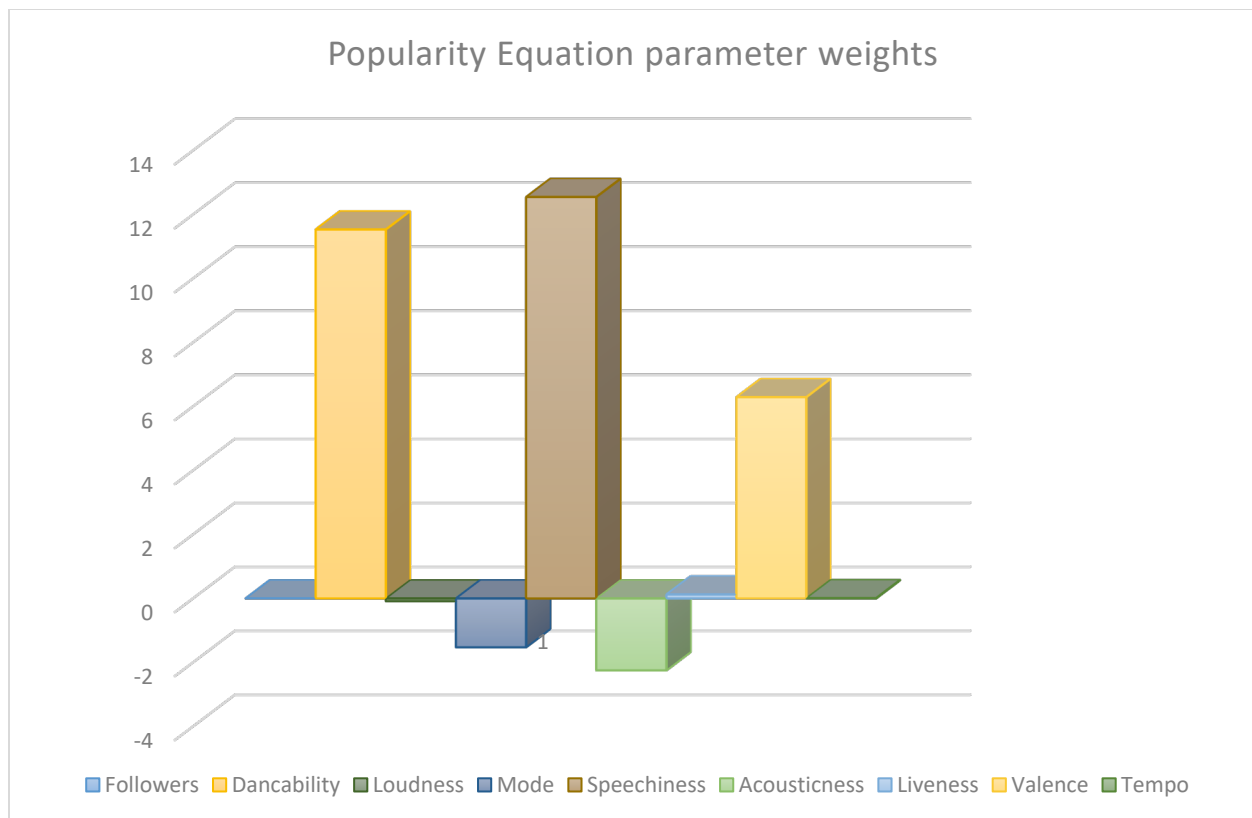
To overcome this limitation, we initially leveraged the power of ChatGPT's brand new beta version of the "code interpreter" plugin (see red circle on next page). (<https://openai.com/blog/chatgpt-plugins>) This enabled us to import the *.csv file that was initially used in Azure directly into ChatGPT. Utilizing the equation with columns described above with almost 10,000 rows, ChatGPT was asked to generate a linear equation using multiple linear regression. These results were promising, and an equation was successfully derived. However, the one found below is not the one delivered by ChatGPT. A Python program was written in VS Code to perform linear regression in the same manner as requested by ChatGPT. Numerous iterations revealed the best solution, which is found below.



The equation produced was:

- **Popularity** = $61.3331 + (3.93 \times 10^{-7}) \times \text{Followers} + (11.5315) \times \text{Danceability} - (0.0847) \times \text{Loudness} - (1.5251) \times \text{Mode} + (12.5477) \times \text{Speechiness} - (2.2467) \times \text{Acousticness} + (0.1258) \times \text{Liveness} - (6.2922) \times \text{Valence} + (0.0101) \times \text{Tempo}$

Popularity	
3.93E-07	Followers
11.5315	Dancability
-0.0847	Loudness
-1.5251	Mode
12.5477	Speechiness
-2.2467	Acousticness
0.1258	Liveness
6.2922	Valence
0.0101	Tempo



Results:

popularity	followers	Danceability	Loudness	Mode	Speechiness	Acousticness	Liveness	Valence	Tempo	popularity_predicted
100	78485332	0.519	-7.673	1	0.0274	0.075	0.132	0.267	78.915	96.14035357
100	78485332	0.354	-13.69	1	0.0281	0.731	0.0858	0.113	94.219	93.93886203
100	78485332	0.602	-5.778	1	0.0338	0.00196	0.105	0.471	96.969	95.61447613
100	78485332	0.624	-2.94	1	0.0296	0.00265	0.189	0.658	121.07	93.42389564
100	78485332	0.777	-6.942	1	0.0522	0.757	0.108	0.172	139.883	97.63948834
100	78485332	0.472	-3.72	1	0.0279	0.091	0.23	0.304	147.854	93.72877677
100	78485332	0.392	-9.195	1	0.17	0.833	0.145	0.529	81.112	93.50487737
100	78485332	0.636	-7.855	1	0.031	0.0494	0.107	0.208	125.952	97.27862442
100	78485332	0.58	-6.462	1	0.0251	0.575	0.121	0.425	76.009	94.87370596
100	78485332	0.316	-10.381	1	0.0488	0.878	0.0797	0.221	74.952	92.77302428
100	78485332	0.71	-6.965	1	0.0366	0.00164	0.0785	0.673	135.012	94.12101318

Phase 2.0 Enhancements

If the group had more time or we had formed an ongoing business, there are numerous enhancements that could be envisioned.

1. Automation – Building a delivery pipeline to and from the equations would be most beneficial and easy to run.
 - a. User input would guide the parameter changes and is much like is shown in Haitie Liu's RShinyApp.

2. Marketing to musicians as a way to craft their songs to obtain higher popularity ratings.

Machine Learning (of which linear regression is a part of) is a power tool and will continue its influence on all citizens lives in the years to come.