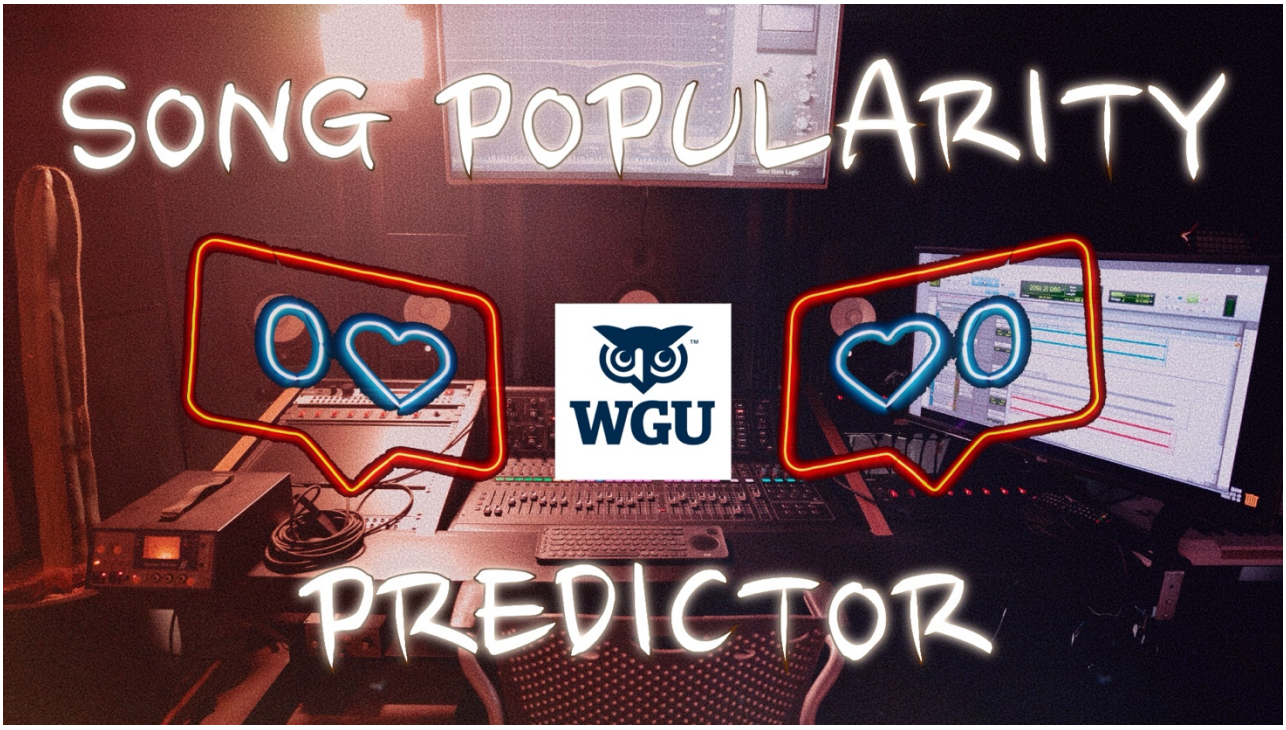
C964 – COMPUTER SCIENCE CAPSTONE

SONG POPULARITY PREDICTOR

Johannes Van Rossum
ID #001527666

WGU EMAIL: JVANROS@WGU.EDU



10/15/2022

TABLE OF CONTENTS

A1. LETTER OF TRANSMITTAL

Johannes Van Rossum
909 E Camelback Rd
Phoenix, AZ 85014

October 17, 2022

SONY Music Entertainment
25 Madison Ave
New York, NY 10010
Attn: Senior Leadership

MUSIC POPULARITY PREDICTOR – SOFTWARE PROPOSAL

Dear Senior Leadership,

"Too Many Songs, Not Enough Hits: Pop Music Is Struggling to Create New Stars" (Leight, 2022) is the eye-catching title of a recent article on Billboard.com. As you may well know from your own experience, building an audience for new acts is harder than ever due to an overflow of new music and songs in the marketplace. The same article goes on to say: "It's a bigger and more level playing field, and everything is getting lost." Over 80,000 tracks are being uploaded to major digital service providers each day and the songs that break seemingly do so by mere luck. Finding what is lost is now more time consuming, requires specialized personnel with unique skillsets, and is therefore more expensive. Machine Learning for Music & Media (MLMM), a company founded by a schooled musician, recording artist, and Machine Learning-engineer (with a Bachelor's degree in Computer Science and independent certifications from CompTIA, ITIL, and EdX), dedicates itself to finding what got lost, to uncovering that gem from the muddied waters, based on the philosophy that true talent will always float to the surface. MLMM will develop for you a data-driven, web-

based solution that will show it is not solely mere luck. The Song Popularity Predictor (SPP) is a tool to help clear the muddied waters and will predict for you a (new) song's popularity based on its extracted musical attributes, such as danceability, energy, tempo, and much more. This application will benefit you in several ways:

- Unparalleled efficiency – sift through 80,000 songs in less than an hour and discover what new songs have the potential to be popular.
- Reduce costs – No more sitting through hours and hours of listening to bad demo tracks, no more wasting time on traveling to obscure nightclubs in all corners of the country, and no more need for a huge team of employees.
- Adaptability – adapting the model to comply with current music trends is easy and can be done at any time.
- Great potential for creative human & AI collaboration - Human perception is highly subjective and may be very different from what the AI thinks. As this field develops, humans can collaborate with AI to find new features that make a song popular, songwriters can use the tool to see whether they are on the right track with their new projects, and you and your artists can use SPP's advice and predictions to enhance the product.

Development, implementation, and maintenance of this revolutionary tool will require an initial investment of $24,000 with a yearly maintenance cost estimated at 20% ($4,800).

Yours sincerely,

Johannes Van Rossum

A2. PROJECT RECOMMENDATION

PROBLEM SUMMARY

With over 80,000 tracks being uploaded to major digital service providers each day the music industry market has become convoluted over the past years. Finding new talent and catchy songs has become more challenging, more time-consuming, and more costly. Machine Learning for Music & Media (MLMM), a company founded by a schooled musician, recording artist, and Machine Learning-engineer, dedicates itself discovering the hidden gems in a convoluted market. One tool that will help in this endeavor is the Song Popularity Predictor (SPP) by predicting a (new) song's popularity based on its extracted musical attributes, such as danceability, energy, tempo, and much more.

APPLICATION BENEFITS

The SPP will benefit both the music industry and individual creatives:

- Unparalleled efficiency – sift through 80,000 songs in less than an hour and discover what new songs have the potential to be popular.
- Reduced costs – Piles of demo tracks on desks belong to the past, there's no need wasting money on unfruitful scouting efforts, and the application will take over a lot of work previously done by humans, implicating a reduction in the need for manpower.
- Great potential for creative human & AI collaboration - Human perception is highly subjective and may be very different from what the AI thinks. As this field develops, humans can collaborate with AI to find new features that make a song popular and to adapt to current musical trends. On a personal level, this tool will enable songwriters to check the quality of their new projects in the light of what the model thinks the current trends are and use SPP's advice to enhance their product.

APPLICATION DESCRIPTION

The application will be a web application dashboard that is freely accessible to anyone in your company worldwide. Its features will be:

- The ability so set 13 different audio features such as duration, tempo, key, and loudness.
- A visual representation of the data input by the user.
- A song popularity prediction, having three possible outcomes: unpopular, popular, and very popular.

In addition, the Jupyter Notebook holding the data that the Machine Learning (ML) model will be trained on will be included. Just as musical trends evolve, the model will have to evolve during future iterations of the project. At the model's core sits a Random Forest algorithm that will predict a song's popularity with at least 80% score on precision, recall, and f1-score. These scores will only increase the more the song database grows. As the database grows, it is important that the data can be understood in a visual way. This can all be done using the Jupyter Notebook, where Python libraries such as Sci-kit Learn, Pandas, Numpy, Plotly, Seaborn, and Matplotlib help with data loading, cleaning, visualizing, feature engineering, preprocessing, and making predictions.

The web application will be built with Streamlit, which turns Python scripts into web applications in minutes.

DATA DESCRIPTION

The dataset used for training and testing the Random Forest model (Spotify and Genius Track Dataset) is publicly available on Kaggle.com. It is updated and maintained periodically. The csv file, loadable using Pandas, can be found and downloaded at the following URL: https://www.kaggle.com/datasets/saurabhshahane/spotgen-music-dataset/download?datasetVersionNumber=328.

OBJECTIVE AND HYPOTHESES

The SPP's objectives are to save time, save costs, and save frustration by taking over tedious tasks. The User Interface (UI) will be intuitive and easy to understand. The UI can be used to input song features and will output a prediction for its popularity with an accuracy and recall of at least 80%: unpopular, popular, or very popular.

If we can find a machine learning classification algorithm that works well as a solution to our problem and the right song attributes are used as input for the predicting model, the model will be able to predict the song popularity with an accuracy and recall score of at least 80% (which is the same as 0.80 in the model's accuracy report).

METHODOLOGY

The CRISP-DM Agile methodology and principles will be applied to the implementation of this project. The codebase will be developed in sprints. That way the application can be improved incrementally as data understanding grows and backtracking is possible when necessary.

1. *Business Understanding:* In this phase, we must work towards an understanding among leadership, executive staff, and other managing staff about why this project is needed. What worked in the last century, doesn't necessarily always work in this one. To adapt to the times and ensure growth for the future, the company needs the help of ML-tools.

2. *Data Understanding:* In this phase, we will collect, analyze, and visualize the data set using a Jupyter Notebook. Visualizing the data will prove useful in understanding the data and identify relationships (if any). For example, is there a relationship between popularity and the level of positivity in a song? Can we identify a relationship between the tempo of a song and its popularity?

3. *Data Preparation:* In this phase, we prepare the final data set for modeling. This will be done by selecting the data we want and cleaning it from data that we don't want. We are solely zooming in on 13 song attributes, namely: acousticness, danceability, duration (in ms), energy, instrumentalness, key, mode, liveness, loudness, speechiness, tempo, time signature, and valence. Even though one can argue an artist name can boost popularity of a song, external factors like that are outside the scope of this project.

4. *Modeling:* In this phase, we will build and assess a classification model (like a Random Forest Classifier) that will help us achieve the application's goals. We will split the dataset 80/20; 80% of the dataset will be used for training the model and 20% will be used for testing.

5. *Evaluation:* In this phase, we will determine how the model meets the goals and objectives. Is the model making correct predictions? Is it ready for deployment or do we need to start another iteration to improve it? Was anything overlooked?

6. *Deployment:* If all previous phases have been completed to satisfaction, we will be ready for deployment. The application will be deployed using Streamlit, making it available in a web browser to all authorized employees anywhere in the world. Feedback will be collected to improve the current working version and ideas will be collected to identify future projects that build on this one.

FUNDING REQUIREMENTS

Please refer the table for initial project funding estimates:

Service	Cost/hour	Total hours	Total cost
Planning and Design	$100	30	$3,000
Development	$200	40	$8,000
Documentation	$150	30	$4,500
Design Review and Determining Future Work	$150	30	$4,500
Overhead Costs (20%)			$4,000
		TOTAL:	$24,000

STAKEHOLDER IMPACT

There are two main groups of stakeholders:

- The music industry
- The creatives (songwriters and composers)

The impact on both groups is a little different, but sometimes related. For the music industry:

- Improved efficiency. More songs can be analyzed in much shorter time.
- Increased profitability by lowering operational costs.
- Faster time to discovery. Discover talent and potential popular songs quicker than the competition.
- Increased popularity success rate. Men lie, women lie, but numbers don't. The ratio of released songs by the label versus the number of actual popular songs will increase.

For creatives:

- Easy and intuitive to use – the SPP is a hands-on tool for suggestions on how to improve a new song.
- Confidence boosting – data backed confirmation to boost confidence levels.
- Education – Explore what makes other songs popular and what doesn't.

DATA PRECAUTIONS

The data itself is publicly available either on Kaggle.com or via the Spotify API. This data is not considered sensitive or protected.

DEVELOPER'S EXPERTISE

Two additional developers will be added to the existing development team that consists of a Python programmer and a machine learning engineer. The current team members combined have over 12 years of experience.
The new members will be required to be strong Python programmers, have experience in web development and Streamlit, be flexible, and comfortable learning new technologies. Experience in deploying applications to services like AWS and Heroku are preferred.

B. TECHNICAL PROPOSAL

PROBLEM STATEMENT

With over 80,000 tracks being uploaded to major digital service providers each day the music industry market has become convoluted over the past years. Finding new talent and catchy songs has become more challenging, more time-consuming, and more costly. The challenge is to utilize Machine Learning as a tool to take on this challenge and to do this work faster, cheaper, and more efficiently than humans ever could. One tool that will help discover the gems in a convoluted market is the Song Popularity Predictor (SPP), a machine learning model that predicts a (new) song's popularity based on its extracted musical attributes, such as danceability, energy, tempo, and much more, through a user-friendly UI in a web browser environment to enable a high level of accessibility.

CUSTOMER SUMMARY

This application will satisfy the business needs of those in the company charged with promoting existing artists, those working to find new, hidden talent and the company's songwriters/composers.

- The app will be a tool to help pick the next song release for an existing artist, simply by comparing the release options' attributes and picking the one with the highest popularity prediction.
- Those with the responsibilities of finding and signing new talent can input the song's attributes into the tool to help them decide who is going to be signed on a record deal next.
- The company's songwriters and composers will use the app to confirm if their latest written songs are going to be popular or not. They'll also be able to identify what areas in the song can be improved to increase its popularity potential.

No special skills are needed to use the application, thanks to the user-friendly UI for which only basic web navigation skills are required.

EXISTING SYSTEM ANALYSIS

Currently, the company has no machine learning tool in place. It only has access to Spotify for Artists, which gives basic information regarding listener's demographics and track performance. What songs to release next is largely based on a hunch and lots of manpower is needed to filter through the more than 80,000 new tracks being uploaded to major distribution platforms each day. Songwriters and composers are only as good as their last song. Upon completion of this project, the next song to release is an educated guess to say the least, employees can filter through new song releases faster than before, and songwriters/composers will have the ability to score their compositions and make adjustments if necessary.

DATA

The dataset used for training and testing the initial model is in .csv format, clean, and publicly available on Kaggle.com. Future data will be collected through the Spotify API, converted to .csv format, and run through the SPP app. At any time can new data be added to the original dataset to retrain and update the prediction model. New data will most likely always be clean, as the model uses the same song attributes that Spotify outputs through their API. Newly acquired data will have to be cleaned from unnecessary attributes before adding to the existing dataset. These attributes are analysis_url, id, track_href, type, and uri.

PROJECT METHODOLOGY

To manage this project, the Agile methodology will be used.

1. Concept: here, we will determine and document the project's scope. After gathering key requirements, documentation will be produced to outline them, including what features will be developed and what the end results should be. Estimated time requirements and costs will be documented as well.

2. Inception: here the design process is started. A UI mockup will be created and project architecture will be built.

3. Iteration: this is the construction phase, where the bulk of the work will be carried out. The design will be turned into code, while taking into consideration all the requirements. At the end of the first sprint, a prototype product will be produced

with minimal functionality. Additional features and adjustments will be added in later iterations. Being able to show incremental improvements will help with client satisfaction.

4. Release: After performing full quality assurance through testing, the product will be ready for deployment. Potential bugs or defects will be addressed swiftly. More documentation will be produced to help with the training of users.

5. Maintenance: The web application will now be fully available to users and the Jupyter Notebook will be ready to be used by those responsible for project maintenance. Our development team will provide ongoing support to resolve any new bugs. When necessary, additional training can be done to ensure all users know how to use the application. Over time, new iterations can be performed to enhance the existing product with additional features, upgrades, and use-cases.

PROJECT OUTCOMES

The project outcomes consist of two distinct categories: Project Deliverables and Product Deliverables.

Project Deliverables:

- Milestones schedule: to keep track of project timeliness and budget an in-depth schedule of the project milestones will be required.
- Test plan: a test plan outlining pre-train tests to catch bugs before running the model, post-train tests to check whether the model performs correctly, and directional tests to check on how changes in input affect the model prediction.
- Pseudo-code: pseudo-code shall be created to outline the data analysis, data preparation, data processing, training, and evaluation phases. The same shall be done for the web application, including a UI mockup.
- Coding Requirements: a list of all environment requirements and Python libraries needed to run the Jupyter Notebook and the web application.

Product Deliverables:

- Jupyter Notebook – a Jupyter Notebook shall be created that analyzes, visualizes, and process the dataset. Finally, it will train explore the performance of several different models, after which the best model will be chosen to be deployed for use in the web application.

- Maintenance tools – the Jupyter Notebook has all the necessary tools to reevaluate and adjust the code to generate better model performance, try new algorithms, and add more attributes to be taken into consideration by the model.
- Web Application – a web application with a user-friendly UI deployed with the help of Streamlit. The user will be able to input 13 different song attributes to generate a popularity prediction.
- Restricted Access optionality – when desired, access to the app can be restricted by only allowing access to people with certain authorized email addresses. The default app setting currently is to grant access to anyone that has the link to the web app. This setting can be adjusted at any time.

IMPLEMENTATION PLAN

The CRISP-DM Agile methodology and principles will be applied to the implementation of this project. The codebase will be developed in sprints. That way the application can be improved incrementally as data understanding grows and backtracking is possible when necessary.

1. *Business Understanding:* In this phase, we must work towards an understanding among leadership, executive staff, and other managing staff about why this project is needed. Techniques and workflows that worked in the last century, don't necessarily work in this one as well. To adapt to the times and ensure growth and sustainability for the future, the company needs the help of ML-tools.
2. *Data Understanding:* In this phase, we will collect, analyze, and visualize the data set using a Jupyter Notebook. Visualizing the data will prove useful in understanding the data and identify relationships (if any). For example, is there a relationship between popularity and the level of positivity in a song? Can we identify a relationship between the tempo of a song and its popularity? To accomplish this, we will use barplots, boxplots, scatterplots, etc. using the seaborn, plotly, and matplotlib libraries.
3. *Data Preparation:* In this phase, we prepare the final data set for modeling. This will be done by selecting the data we want and cleaning it from data that we don't want. We are solely zooming in on 13 song attributes, namely: acousticness, danceability, duration (in ms), energy, instrumentalness, key, mode, liveness, loudness, speechiness, tempo, time signature, and valence. Even though one can

argue an artist name can boost popularity of a song, external factors like that are outside the scope of this project.

4. *Modeling:* In this phase, we will build and assess several classification models (like Logistic Regression, Random Forest Classifier, and boosting models like Adaboost) that will help us achieve the application's goals. All models are examples of supervised learning. We will split the dataset 80/20; 80% of the dataset will be used for training the model and 20% will be used for testing.

5. *Evaluation:* In this phase, we will compare models and determine which one performed the most desirable by looking closely at the confusion matrices and accuracy reports. Then we will decide if the chosen model is ready for deployment. Perhaps another iteration is needed to improve it. These improvements can be achieved by doing better data resampling, more feature engineering, trying different encoding techniques, or doing more transformations such as log transformation.

6. *Deployment:* If all previous phases have been completed to satisfaction, we will be ready for deployment. The application will be deployed using Streamlit, making it available in a web browser to all authorized employees anywhere in the world. Feedback will be collected to improve the current working version and ideas will be collected to identify future projects for improvement that build on this one.

EVALUATION PLAN

Model accuracy validation and verification is done through ML accuracy reports and by using confusion matrices. Throughout the development process, the Jupyter Notebook and web application will be subject to continuous testing and evaluation, enforced by the Agile development process, which allows us to resolve issues and course correct quickly when issues are discovered.

RESOURCES AND COSTS

PROGRAMMING ENVIRONMENT

The tools and environment needed for this project are:

- Any computer or device able to run a web browser in order to use the web application.
- Jupyter Notebook (including software like Anaconda)
- Python (version 3.9 at minimum), including all required libraries (Numpy, Pandas, Seaborn, Matplotlib, ipywidgets, pickle, imblearn, and sci-kit learn)
- Git version control

ENVIRONMENT COSTS

Resource	Cost
2 additional PC's for newly hired developers	$1,000
TOTAL:	$1,000

HUMAN RESOURCE REQUIREMENTS

Service	Cost/hour	Total hours	Total cost
Planning and Design	$100	30	$3,000
Development	$200	40	$8,000
Documentation	$150	30	$4,500
Design Review and Determining Future Work	$150	30	$4,500
Overhead Costs (20%)			$4,000
		TOTAL:	$24,000

TIMELINE AND MILESTONES

Sprint	Start	End	Tasks
1	Date 10/18/2022 10/24/2022 10/27/2022	Date 10/21/2022 10/26/2022 10/28/2022	**Planning and project setup:** • Define task and scope requirements. • Produce documentation concerning application features and requirements. • Document time requirements and estimates.
2	Date 10/31/2022 11/2/2022 11/7/2022 11/10/2022	Date 11/1/2022 11/4/2022 11/9/2022 11/11/2022	**Data collection, preparation, and visualization:** • Document data surface properties. • Identify data relationships. • Prepare data set for modeling. • Revisit Sprint 1.
3	Date 11/14/2022 11/16/2022 11/18/2022 11/23/2022 12/05/2022	Date 11/15/2022 11/17/2022 11/22/2022 12/02/2022 12/09/2022	**Model training, testing, and validation:** • Build the different ML models. • Train all the ML models. • Test all the ML models. • Revisit Sprint 1 & 2. Complete and finalize all deliverables.
4	Date 12/12/2022 12/15/2022 12/20/2022	Date 12/14/2022 12/19/2022 12/23/2022	**Evaluation:** • Train user on how to use the Jupyter Notebook and web application. • Use and test the system to see if it meets business requirements. Determine the next steps. Revisit Sprint 1, 2, & 3 if necessary.
5	Date 12/27/2022 01/02/2023 01/09/2023	Date 12/30/2022 01/06/2023 ongoing	**Deployment:** • Produce final report and presentation. • Produce a full project review. Monitoring and Maintenance.

C. APPLICATION FILES

Please refer to the following Jupyter Notebook file and web app Python script for the project code:

- C964 - Song Popularity Predictor.ipynb (also viewable at https://github.com/jonivanrossum/C964/blob/main/C964%20-%20Song%20Popularity%20Predictor.ipynb)
  OR
- C964 - Song Popularity Predictor.pdf (also viewable at https://github.com/jonivanrossum/C964/blob/main/C964%20-%20Song%20Popularity%20Predictor%20-%20JupyterLab.pdf)
- webapp.py (also viewable at https://github.com/jonivanrossum/C964/blob/main/webapp.py)

Try out the web app at:

- https://jonivanrossum-c964-webapp/-0yn76e.streamlitapp.com/

D. POST-IMPLEMENTATION REPORT

PROJECT PURPOSE

With over 80,000 tracks being uploaded to major digital service providers each day, the music industry market has become convoluted over the past years. Finding new talent and catchy songs has become more challenging, more time-consuming, and more costly. This machine learning application was developed to satisfy the business needs of those in the company charged with promoting existing artists, those working to find new, hidden talent and the company's songwriters/composers.

- The app is a tool to help pick the next song release for an existing artist, simply by comparing the release options' attributes and picking the one with the highest popularity prediction.
- Those with the responsibility of finding and signing new talent can input a song's attributes into the web application to help them decide who is going to be signed on a record deal next.

- The company's songwriters and composers use the app to confirm if their latest written songs are going to be popular or not. They're also able to identify what areas in the song can be improved to increase its popularity potential.

No special skills are needed to use the application, thanks to the user-friendly UI for which only basic web navigation skills are required.

DATASETS

Let's start off with an example of the dataset in its original form by showing its columns:

```
Columns present in track_data: Index(['Unnamed: 0', 'acousticness', 'album_id', 'analysis_url', 'artists_id',
       'available_markets', 'country', 'danceability', 'disc_number',
       'duration_ms', 'energy', 'href', 'id', 'instrumentalness', 'key',
       'liveness', 'loudness', 'lyrics', 'mode', 'name', 'playlist',
       'popularity', 'preview_url', 'speechiness', 'tempo', 'time_signature',
       'track_href', 'track_name_prev', 'track_number', 'uri', 'valence',
       'type'],
      dtype='object')
```

Figure 1: example of columns present in track_data

Here is an example of the dataset in its final form:

	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	mode	speechiness	tempo	time_signature	valence	pop_rating
0	0.1010	0.748	237667.0	0.666	0.000653	6.0	0.0976	-6.094	0.0	0.0833	114.982	4.0	0.359	Unpopular
1	0.1910	0.608	243667.0	0.664	0.042700	5.0	0.1200	-8.261	0.0	0.0435	100.011	4.0	0.513	Unpopular
2	0.0786	0.470	157500.0	0.828	0.000000	9.0	0.1780	-6.280	1.0	0.0700	96.149	4.0	0.856	Popular

Figure 2: example of columns present in final dataset

The original dataset had so many columns that trying to call tracks_data.head() resulted in a table that fell off the page. It contained many columns that are of no importance to whether a song is popular or not, such as album_id, analysis_url, available_markets, and disc_number. Here is a code example where unnecessary columns were dropped, resulting in a DataFrame we could analyze and process:

```
# Let's drop columns we don't need for our predictor
data = song_data.drop(['Unnamed: 0',
                       'album_id',
                       'analysis_url',
                       'artists_id',
                       'available_markets',
                       'country',
                       'disc_number',
                       'href',
                       'id',
                       'lyrics',
                       'name',
                       'playlist',
                       'track_href',
                       'preview_url',
                       'track_href',
                       'track_name_prev',
                       'track_number',
                       'uri',
                       'type',], axis=1)

data.columns
```

Figure 3: dropping unnecessary columns from the dataset

Next, we wanted to analyze popularity trends over the years and so we needed the release_date column from the album_data DataFrame. To achieve that, we joined song_data and album_data together on track_id.

```
# now we'll merge song_data with the album data release date
song_data = pd.merge(song_data, album_data[['track_id', 'release_date']], left_on='id', right_on='track_id', how='inner'

# drop 'track_id' column again, because we don't need it anymore.
song_data.drop('track_id', axis=1, inplace=True)

# check song_data head
song_data.head()
```

Figure 4: merging two pandas DataFrames

Then, we extracted release_year from release_date, added it to the DataFrame as a column, and dropped the release_date column because it was no longer needed.

```
# create a new column 'release_year' and drop 'release_date'
data['release_date'] = pd.to_datetime(data['release_date'])
data['release_year'] = data.release_date.dt.year
data.drop('release_date', axis=1, inplace=True)
data.head(2)
```

ms	energy	instrumentalness	key	liveness	loudness	mode	popularity	speechiness	tempo	time_signature	valence	genres	release_year
0.0	0.308	0.000000	6.0	0.2530	-10.340	1.0	31.0	0.9220	115.075	3.0	0.589	[]	2011
7.0	0.666	0.000653	6.0	0.0976	-6.094	0.0	47.0	0.0833	114.982	4.0	0.359	['electra']	2018

Figure 5: creating the new column release_year

As a last example, we performed binning on the popularity column. This problem is more a classification problem than a regression problem, so we did some research and found that a song with a popularity rating of at least 50.0 is considered popular by the Spotify API from which this data was extracted. All records with popularity rating 0.0 were deleted from the DataFrame, since we're interested in what makes a song popular (not what makes it unpopular necessarily). All records with a rating of higher than 0.0, but lower than 50.0 was rated as "unpopular". We then decided to split the popular section up in 2 subsections: "popular" and "very popular". With "very popular" starting at a rating of at least 80.0. The company is of course interested in all potentially popular music, but especially in those songs that have the potential to become *very* popular! Here's the code snippet:

```
df['pop_rating'] = ''
```

```
pop_ratings = ['Unpopular', 'Popular', 'Very Popular']

for i, row in df.iterrows():
    score = pop_ratings[0]
    if (row.popularity >= 80):
        score = pop_ratings[2]
    elif (row.popularity >= 50 and row.popularity < 80):
        score = pop_ratings[1]
    df.at[i, 'pop_rating'] = score

df[['popularity', 'pop_rating']].head()
```

	popularity	pop_rating
1	47.0	Unpopular
2	35.0	Unpopular
4	55.0	Popular
6	41.0	Unpopular
7	49.0	Unpopular

Figure 6: replacing ordinal values

Other than that, the dataset did not have null or NaN values, so there was no need for imputing, for example.

DATA PRODUCT CODE

For the full product code, please refer to the following Jupyter Notebook file and web app Python script:

- C964 - Song Popularity Predictor.ipynb (also viewable at https://github.com/jonivanrossum/C964/blob/main/C964%20-%20Song%20Popularity%20Predictor.ipynb)
  OR
- C964 - Song Popularity Predictor.pdf (also viewable at https://github.com/jonivanrossum/C964/blob/main/C964%20-%20Song%20Popularity%20Predictor%20-%20JupyterLab.pdf)
- webapp.py (also viewable at https://github.com/jonivanrossum/C964/blob/main/webapp.py)

SUMMARIZED FINAL PRODUCT CODE

1. First, all necessary Python libraries were loaded, such as pandas, seaborn, matplotlib, and pickle. We also imported functions such as confusion_matrix, GridSearchCV, and train_test_split from the Scikit-learn library, from which we imported our machine learning algorithms that we wanted to test as well, such as RandomForestClassifier, GradientBoostingClassifier, and BaggingClassifier.

2. We then loaded our three .csv files in a DataFrame using pandas.

3. After exploring the dataset, we dropped the columns we don't need and created useful columns inferred from the existing data.

4. Next, we took a deep dive into the dataset and created an interactive function to analyze popularity trends over the years. The next graph shows the interactive function at work, from which we can clearly conclude that songs have gotten louder over the past 100 years:

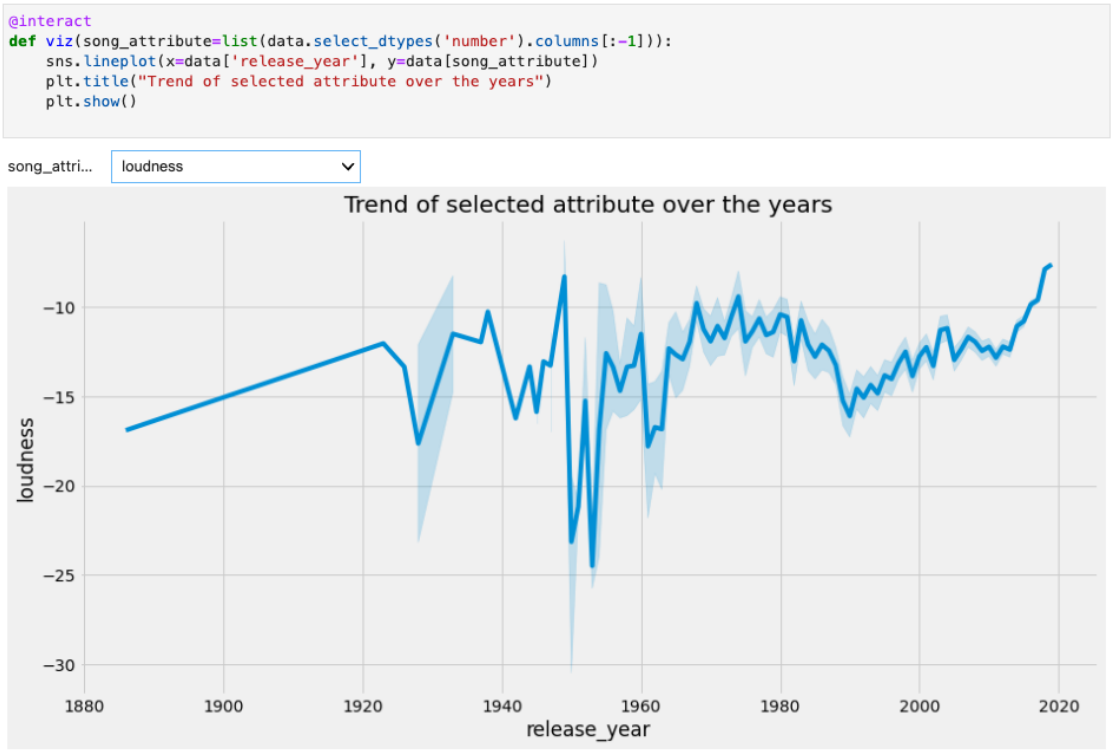


Figure 7: trend of average song loudness over the years

5. Based on exploring the dataset using the interactive function above, we concluded that time is an important factor in song popularity. We therefore decided to drop all records older than the year 2015 from the dataset. We set a YEAR_THRESHOLD variable on the top of the Jupyter Notebook, so that the threshold can be changed in future iterations of the project, if needed.

6. Because this is more a classification problem than a regression problem, we binned the popularity column using three categories: “unpopular”, “popular”, and “very popular” (see Figure 6).

7. We then created another interactive function to do some in-depth bivariate analysis and explored how our chosen song attributes relate to popularity. The next graph shows, for example, that the very popular songs clearly have higher danceability than the less popular ones:

```
song_attribute=list(df.select_dtypes('number').columns[:-1])
song_attribute.remove('popularity')

@interact
def viz2(song_attribute=song_attribute):
    sns.barplot(x=df['release_year'], y=df[song_attribute], hue = df['pop_rating'], palette='Reds')
    plt.title("Song Attribute vs. Popularity")
    plt.show()
```

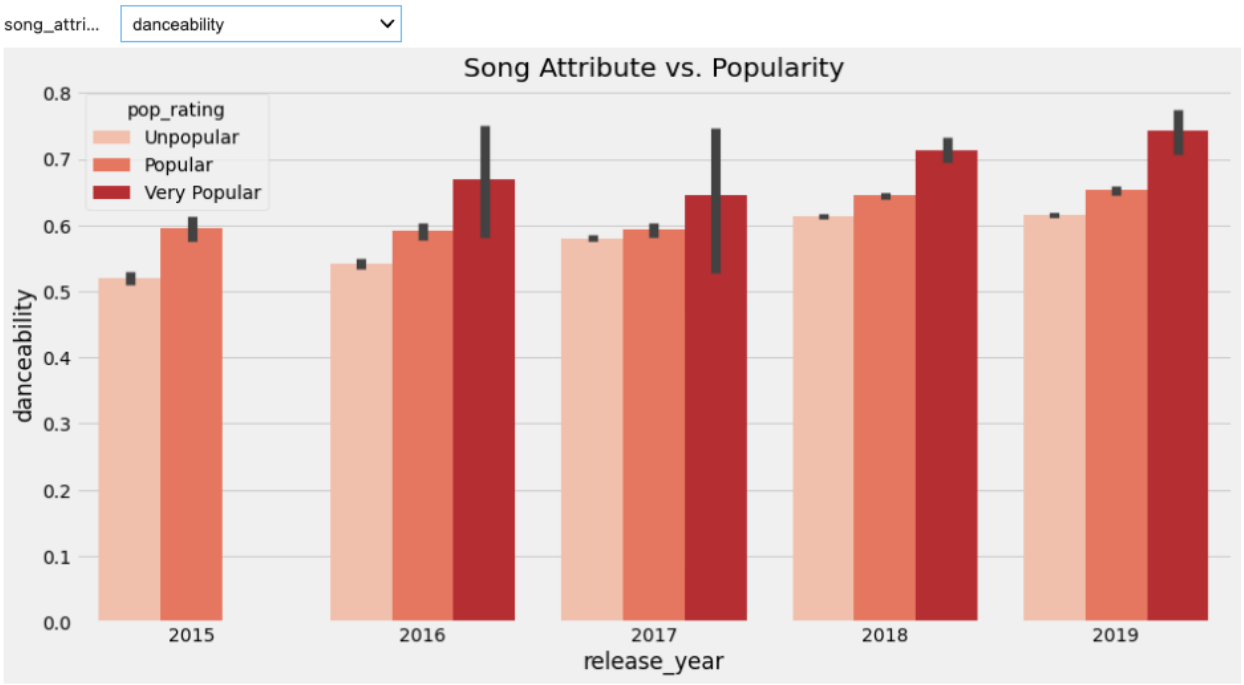


Figure 8: example of how danceability's relation to song popularity

8. As one of several non-descriptive methods used in this project, we wrote a function to plot a regression line in a scatterplot that lays out loudness versus popularity. Even though the data is very scattered, we can see a light trend upwards as a song's loudness level increases.

```
regress_plot(df.loudness, df.popularity, data=df, ylabel="Popularity",
        xlabel="Loudness", title="Loudness vs. Popularity")
```

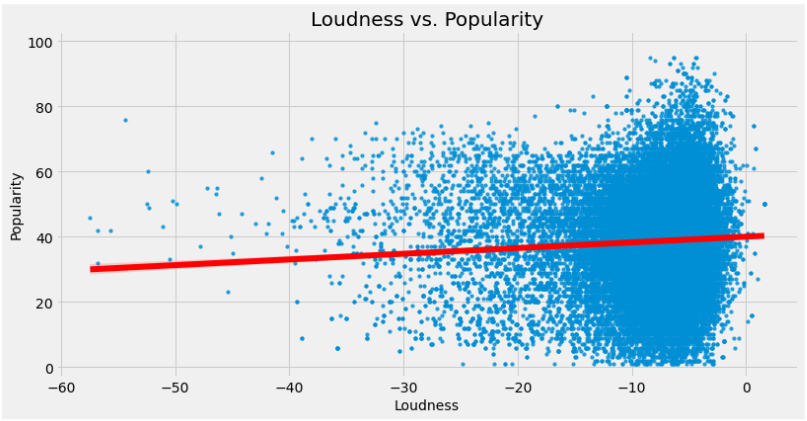
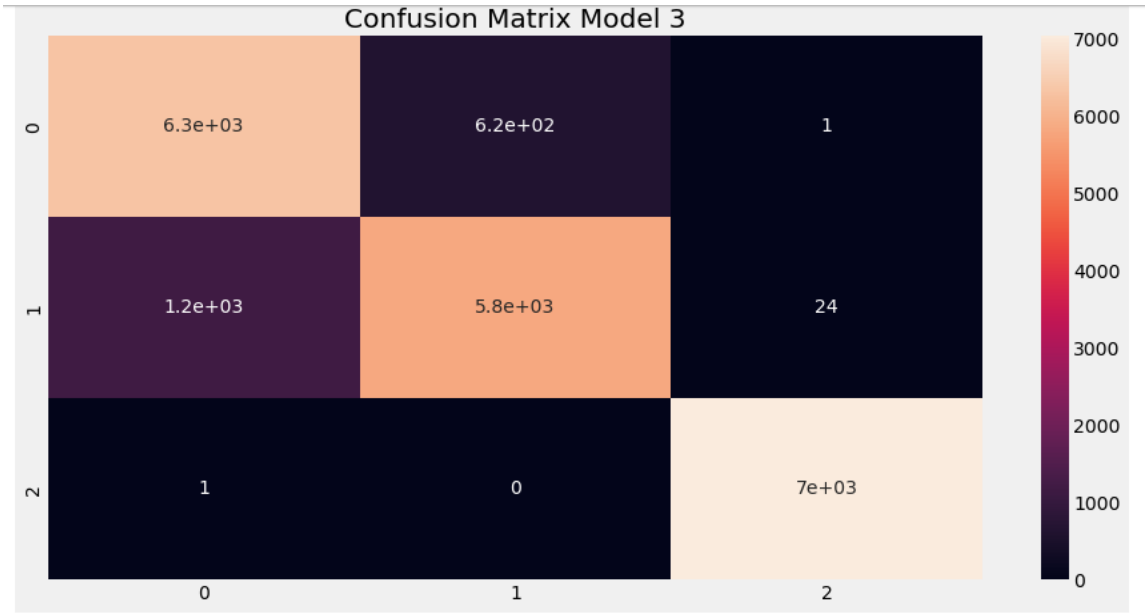


Figure 9: custom regression plot

9. Now that we understood our data better, we were ready to split our dataset in a train set and a test set, using sckikit-learn's train_test_split() function.

10. The target column turned out to be heavily imbalanced with, for example, 27,969 records falling in the "unpopular" category and only 188 in the "very popular" category. So, we used imblearn's SMOTE algorithm to perform oversampling.

11. We then built, trained, and tested several models, after which we concluded that the Random Forest model clearly performed the best. We used GridSearchCV to find the best parameters for the model and trained/tested it again, with amazing results:




```
CPU times: user 50.5 s, sys: 323 ms, total: 50.8 s
Wall time: 51.2 s
```

```
# Accuracy and Classification Report
print(accuracy_score(y_test, y_pred))
print()
print(classification_report(y_test, y_pred))
```

```
0.9140963912856939

              precision    recall  f1-score   support

           1       0.85      0.91      0.88      6934
           2       0.90      0.83      0.87      6997
           3       1.00      1.00      1.00      7046

    accuracy                           0.91     20977
   macro avg       0.92      0.91      0.91     20977
weighted avg       0.92      0.91      0.91     20977
```

Figure 10: confusion matrix & classification report

12. Finally, we saved the model as a pickle file, uploaded it to GitHub using Git LFS, and created a web application file (webapp.py). The application was deployed using Streamlit. It can be visited and tested at: https://jonivanrossum-c964-webapp-0yn76e.streamlitapp.com/

HYPOTHESIS VERIFICATION

This project's hypothesis was established as follows: "If we can find a machine learning classification algorithm that works well as a solution to our problem and the right song attributes are used as input for the predicting model, the model will be able to predict the song popularity with an accuracy and recall score of at least 80%"

As figure 10 shows, we ended up well above our stated target score of 80% with a precision score of 92% and recall at 91%, respectively. Thus, we concluded our hypothesis was accepted.

EFFECTIVE VISUALIZATIONS AND REPORTING

We generously used visualizations and reporting in our Jupyter Notebook to better understand the data, discover trends, and decide on what kind of data manipulation was needed to make this project a success. Here are some examples:

- Firstly, just being able to call the .head() function of pandas immediately informed us what kind of data we were dealing with and what data types were used.
- Interactive visualizations as shown in figure 11 really helped understand the difference in song attributes over the last 100+ years. In this example, we can clearly conclude from the line plot that songs from the 1920's were almost completely acoustic, as opposed to 2019 where songs in general have a very low level of acousticness. This confirmed for us that for our project we should only

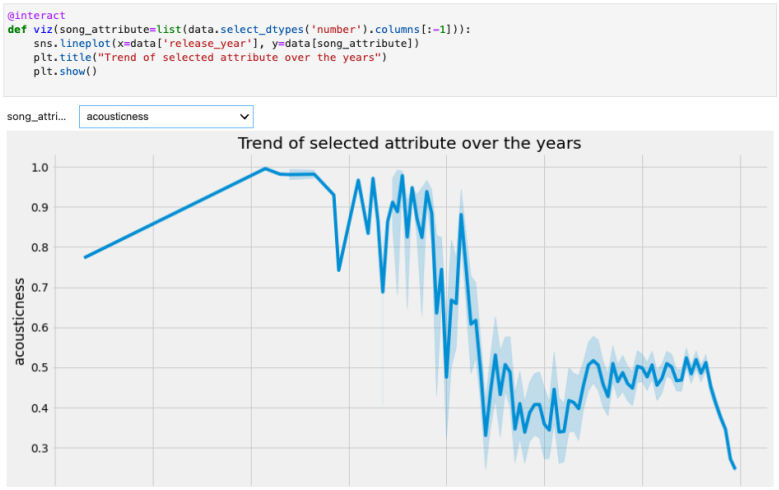


Figure 11: Trend of song acousticness over the years

use data from 2015 on; what was popular in the 1920's is not necessarily popular nowadays.

- The use of heatmaps and scatterplots helped us realize there is (almost) no clear linear relation between a particular song attribute and its popularity score, as shown in figure 12.

```
plt.rcParams['figure.figsize'] = small_figsize
sns.heatmap(df.corr(), annot=True, cmap='copper')
plt.title("Correlation of Music Attributes")
plt.show()
```

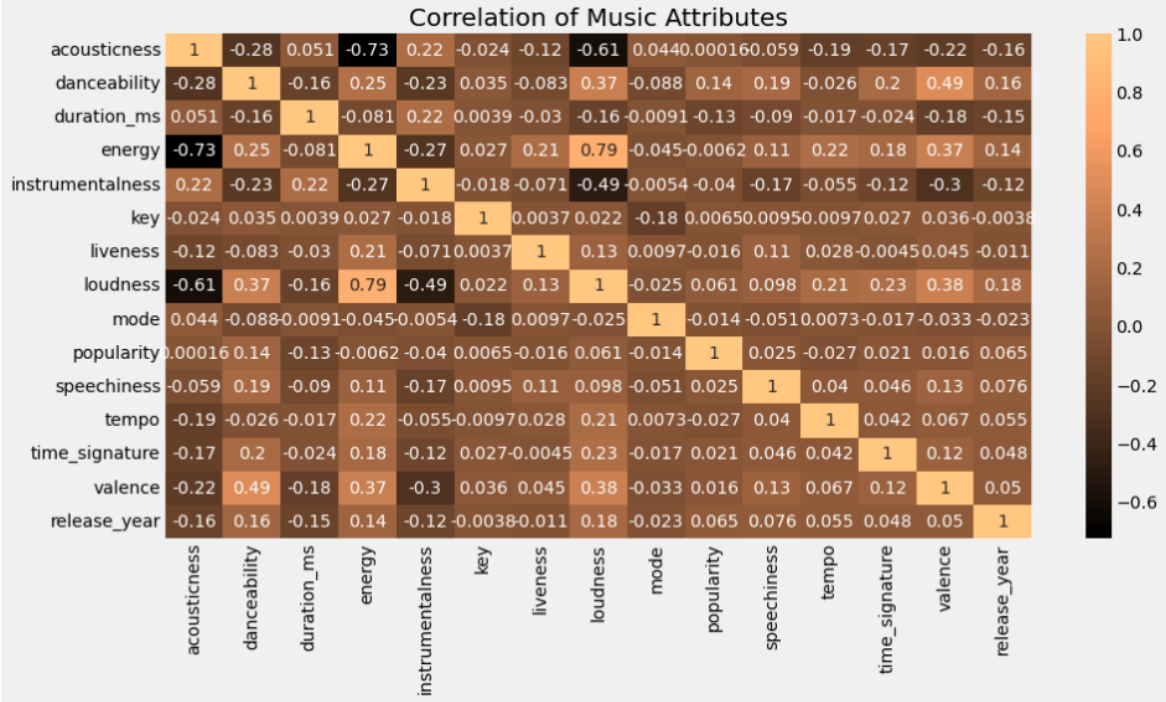


Figure 12: correlation heatmap of song attributes

- Since the correlation heatmap did not give us much to go on, we used bar plots in an interactive visualization to display the relation between popularity and song attributes as well (figure 13). This particular example showed us that, generally speaking, popular and very popular songs have a lower level of instrumentalness than unpopular songs.

```
song_attribute=list(df.select_dtypes('number').columns[:-1])
song_attribute.remove('popularity')

@interact
def viz2(song_attribute=song_attribute):
    sns.barplot(x=df['release_year'], y=df[song_attribute], hue = df['pop_rating'], palette='Reds')
    plt.title("Song Attribute vs. Popularity")
    plt.show()
```

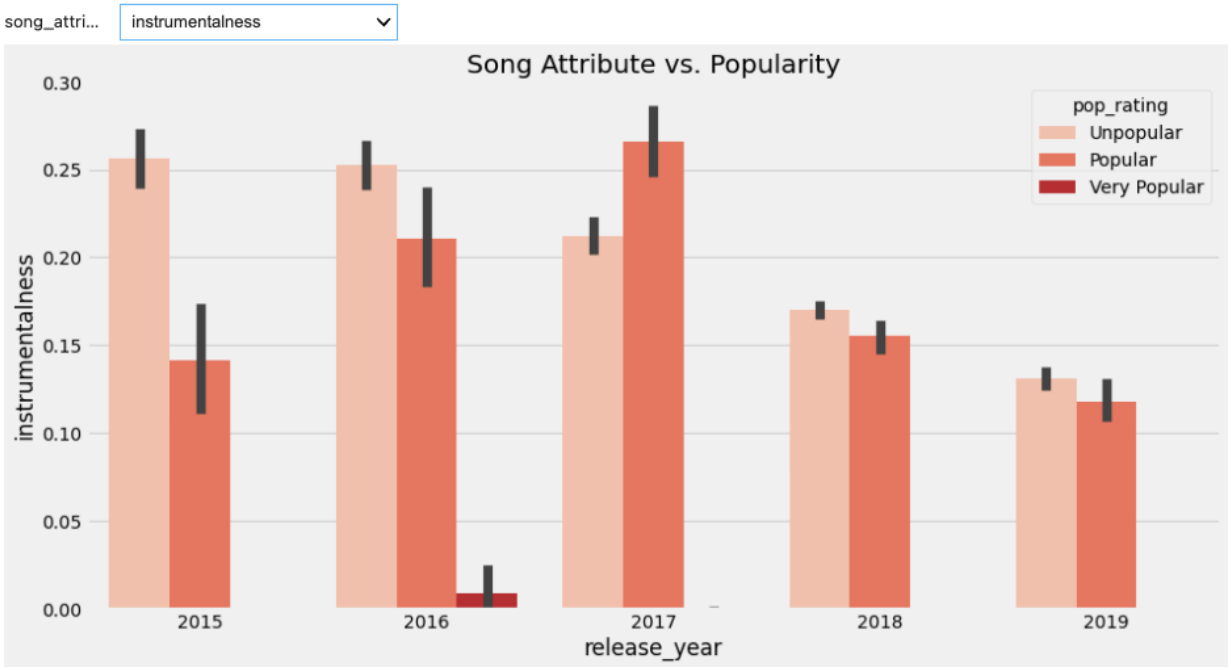


Figure 13: level of instrumentalness laid out against popularity rating

- Scatterplots, like the one shown in figure 14, helped us determine that we needed to transform the popularity score into a popularity rating, so that the problem we are trying to solve became a classification problem and the model would become more accurate in its predictions. After all, we did try out a linear regression model and the results were very poor.

```
scat_plot(df.energy, df.popularity, hue=df.pop_rating,
          xlabel="Energy",
          ylabel="Popularity",
          title="Energy vs. Popularity")
```

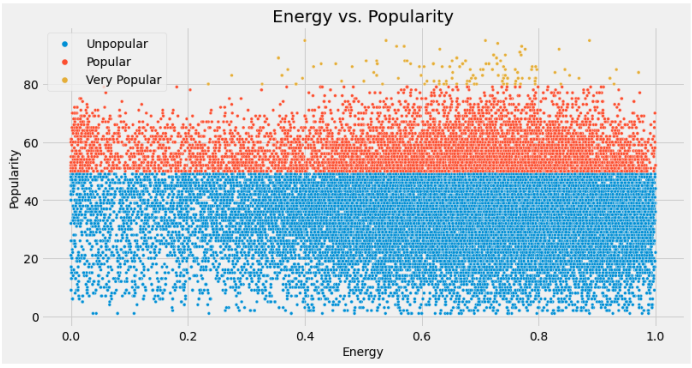


Figure 14: energy vs. popularity score scatterplot

- Plotting histograms showed us not only how song data was distributed, but most importantly showed us that our target column was heavily imbalanced (figure 15). This led to the decision to use over sampling methods (specifically SMOTE) to help our model learn better.

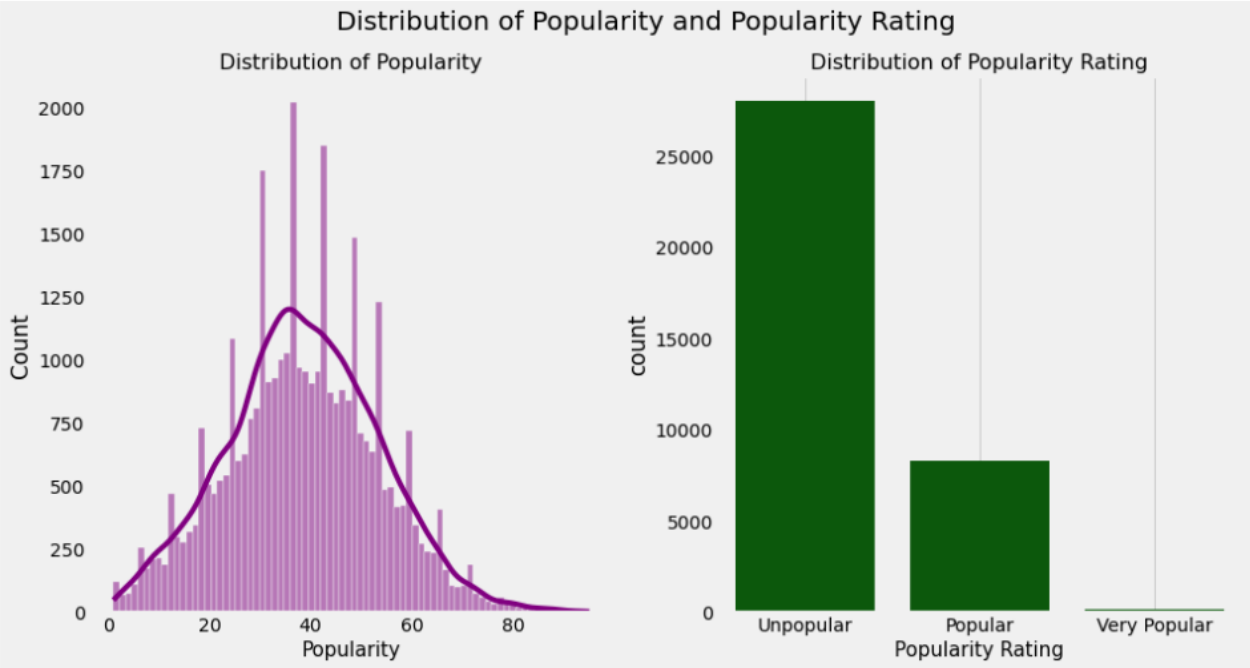


Figure 15: distribution of our target column

- As a last example of how we summarized data, we created some simple reporting functions to get insights about the dataset's mean values (figure 16).

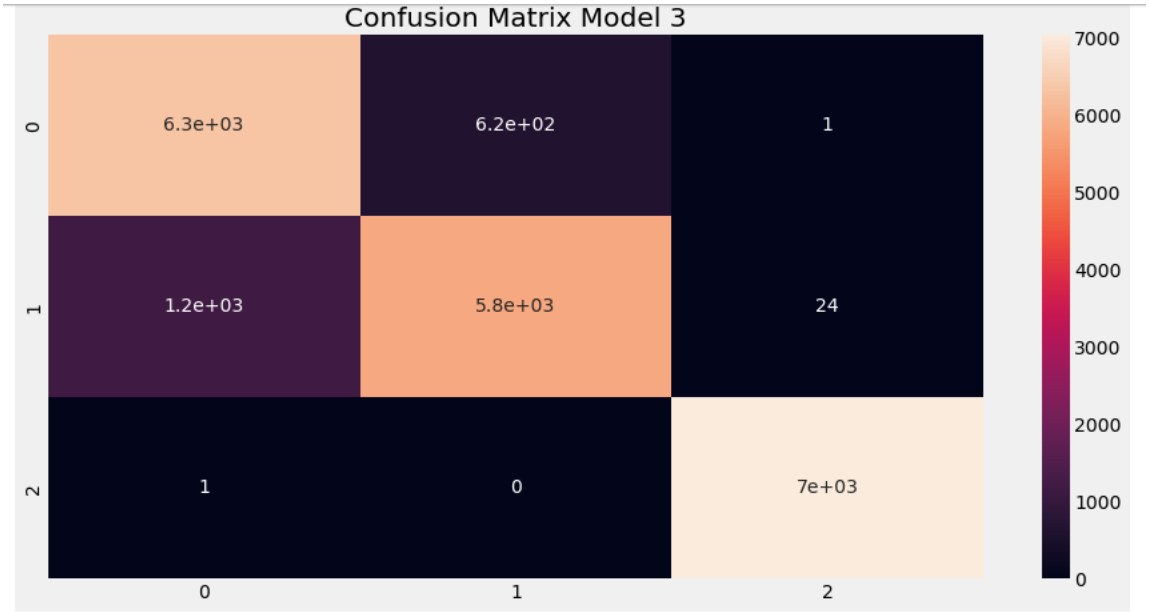
```
print("Average Ratio of Acousticness in the Dataset: {0:.2f}".format(df['acousticness'].mean()))
print("Average Ratio of Danceability in the Dataset: {0:.2f}".format(df['danceability'].mean()))
minutes = int((df['duration_ms'].mean()//1000)//60)
seconds = int((df['duration_ms'].mean()//1000)%60)
print("Average Duration of Songs:", minutes, "minutes and", seconds, "seconds")
print("Average Ratio of Energy in the Dataset: {0:.2f}".format(df['energy'].mean()))
print("Average Ratio of Instrumentalness in the Dataset: {0:.2f}".format(df['instrumentalness'].mean()))
print("Average Ratio of Liveness in the Dataset: {0:.2f}".format(df['liveness'].mean()))
print("Average Ratio of Loudness in the Dataset: {0:.2f}".format(df['loudness'].mean()))
print("Average Ratio of Speechiness in the Dataset: {0:.2f}".format(df['speechiness'].mean()))
print("Average Tempo in the Dataset: {0:.2f}".format(df['tempo'].mean()), "bpm")
print("Average Ratio of Valence in the Dataset: {0:.2f}".format(df['valence'].mean()))
print("Average Popularity in the Dataset: {0:.2f}".format(df['popularity'].mean()))
```

```
Average Ratio of Acousticness in the Dataset: 0.29
Average Ratio of Danceability in the Dataset: 0.60
Average Duration of Songs: 3 minutes and 53 seconds
Average Ratio of Energy in the Dataset: 0.62
Average Ratio of Instrumentalness in the Dataset: 0.18
Average Ratio of Liveness in the Dataset: 0.18
Average Ratio of Loudness in the Dataset: -8.42
Average Ratio of Speechiness in the Dataset: 0.10
Average Tempo in the Dataset: 120.22 bpm
Average Ratio of Valence in the Dataset: 0.46
Average Popularity in the Dataset: 38.55
```

Figure 16: data summary of dataset's mean values

ACCURACY ANALYSIS

With the implementation of Random Forest, using SMOTE for oversampling, and GridSearchCV to determine the best model parameters, the model predicted a song's popularity rating with a 91% accuracy score. This was well above our target score of 80%. A 100% precision and recall score for predicting very popular songs is especially promising.




```
CPU times: user 50.5 s, sys: 323 ms, total: 50.8 s
Wall time: 51.2 s
```

```
# Accuracy and Classification Report
print(accuracy_score(y_test, y_pred))
print()
print(classification_report(y_test, y_pred))
```

```
0.9140963912856939

              precision    recall  f1-score   support

           1       0.85      0.91      0.88      6934
           2       0.90      0.83      0.87      6997
           3       1.00      1.00      1.00      7046

    accuracy                           0.91     20977
   macro avg       0.92      0.91      0.91     20977
weighted avg       0.92      0.91      0.91     20977
```

Figure 17: our model's confusion matrix and classification report

APPLICATION TESTING

We tested our application in different ways:

- Due to the Agile nature of this project, unit and integration testing was performed at every stage of the project.
- We tested several machine learning models to determine which one would solve our problem in the best way (Linear Regression, Decision Tree, Random Forest, AdaBoost, and Gradient Boosting).
- Acceptance testing was performed to verify the accuracy of the popularity predictions.
- System testing was done to confirm the system worked as expected by, for example, checking the integrity of the GitHub repository and availability of the application, deployed using Streamlit, at random times during day over a week period.
- Usability testing (black box testing) was performed on the web application as part of the acceptance testing of the application by both music professionals and "regular" potential users.

APPLICATION FILES

The following files are needed for a successful execution of this application:

File name:	**Description:**	**Viewable at:**
C964 - Song Popularity Predictor.ipynb	Contains all project code, including all data visualizations, the testing and selecting of the right model, and the creation of the pickle file needed to run the web application.	https://github.com/jonivanrossum/C964/blob/main/C964%20-%20Song%20Popularity%20Predictor.ipynb
webapp.py	Contains source code for the web app UI and uses the model's pickle file to make song popularity predictions every time the user changes input parameters.	https://github.com/jonivanrossum/C964/blob/main/webapp.py

spotify_tracks.csv	This csv file contains all Spotify data concerning songs. However, the Jupyter Notebook downloads this dataset automatically.	https://github.com/jonivanrossum/C964-Data-Sources/blob/main/spotify_tracks.csv
spotify_artists.csv	This csv file contains all Spotify data concerning artists. However, the Jupyter Notebook downloads this dataset automatically.	https://github.com/jonivanrossum/C964-Data-Sources/blob/main/spotify_artists.csv
spotify_albums.csv	This csv file contains all Spotify data concerning song albums. However, the Jupyter Notebook downloads this dataset automatically.	https://github.com/jonivanrossum/C964-Data-Sources/blob/main/spotify_albums.csv

USER'S GUIDE

JUPYTER NOTEBOOK INSTRUCTIONS

Python 3.9 is a minimum requirement if you want to run any of the code locally on your machine. You can download the latest version for your OS at:

https://www.python.org/downloads/

The Jupyter Notebook walks you through the machine learning model's development process. To view the Notebook project code, click the following URL:

https://github.com/jonivanrossum/C964/blob/main/C964%20-%20Song%20Popularity%20Predictor.ipynb

or refer to the file "C964 - Song Popularity Predictor.ipynb" attached to this submission.

You can run all the cells locally if you'd like; the datasets are automatically downloaded from GitHub and loaded using pandas. Just be aware that towards the end of the Notebook a pickle file is saved, and you may not want that. Be on the lookout for this cell; the warning is written in a comment and looks like this:

```
# Save the Random Forest model in a pickle file
# Don't run this cell for evaluation purposes,
# as it'll save a pickle file to your hard drive, and you might not want that.
filepath = 'music_pop_predictor.pkl'
with open(filepath, 'wb') as f:
    pickle.dump(model3, f)
```

If you don't have Jupyter Notebook installed, you can install it with `pip`:

```
pip install jupyterlab
```

Once installed, launch JupyterLab with:

```
jupyter-lab
```

Then navigate to the folder where you downloaded the Notebook file and run it by double-clicking.

If any of the required libraries aren't installed on your machine, just use the top cell in the Notebook to install it. Type: "!pip install" (without the quotation marks, but don't forget the exclamation mark) followed by the name of the library. For example:

```
!pip install pandas
```

You can also use a command line terminal to install Python libraries by typing, for example:

```
pip install sklearn
```

WEB APPLICATION INSTRUCTIONS

You can access the Streamlit web application by clicking the following URL:

https://jonivanrossum-c964-webapp-0yn76e.streamlitapp.com/

To gain access to the application, type "username" as a username and "password" as the password.

Have fun and play around with the user input parameters! Here are some parameter combinations for you to try out and observe different popularity rating predictions:

	Unpopular	**Popular**	**Very Popular**
Acousticness	0.32	0.08	0.03
Danceability	0.34	0.47	0.73
Duration_ms	207345.0	157500.0	220454.0
Energy	0.86	0.83	0.79
Instrumentalness	0.0	0.0	0.0
Key	11	9	0
Mode	1	1	0
Liveness	0.22	0.18	0.11
Loudness	-3.27	-6.28	-5.13
Speechiness	0.10	0.07	0.05
Tempo	179.14	96.15	140.0
Time_Signature	4.0	4.0	4.0
Valence	0.79	0.86	0.36

SUMMATION OF LEARNING EXPERIENCE

This project really brought together all the skills I acquired through the classes at WGU and presented some unique challenges. Introduction to Artificial Intelligence and Project Management already sparked something in me and while working through the challenges of this Capstone Project, I started realizing that this is the field I want to build a career in: Music & Machine Learning.

With the help of WGU resources, several Python libraries documentation, Udemy, GitHub, and Google I was able to bring this project to a successful end. Even though the path was not always clear and despite feeling overwhelmed at times, this learning experience proved again that where there is a will, there is a way. It put me on paths

I did not imagine taking before and introduced me to the possibilities of machine learning and composing music, for example.

I am thankful for this learning experience, and I feel confident I will find a career path that suits my unique skill set and abilities.

E. SOURCES

Leight, E. (October 11, 2022*). Too Many Songs, Not Enough Hits: Pop Music Is Struggling to Create New Stars.* Billboard.com. Retrieved from: https://www.billboard.com/pro/new-music-tiktok-artist-development-suffering/