We were tasked with making an ETL pipeline for Kickstarter Campaigns in order to analyze what makes a successful campaign. To do this, we extracted and transformed the data from Excel files using Python, created CSV files from that transformed data, and used those CSV files to create an entity relationship diagram and table schema, which we loaded into a PostgreSQL database for further querying and analysis.

Data Extraction & Transformation

Given an Excel file with information regarding crowdfunding campaigns, we were tasked with creating separate DataFrames to organize the data and eventually create a database. We accomplished this using the pandas and numpy libraries in Python code.

After reading the file into a pandas DataFrame, we could see that the category and subcategory for each campaign were together in the same column. We separated them by splitting the values on the "/", created two new columns, "category" and "subcategory," and dropped the original combined column.

We created lists for each unique value in both the category and subcategory columns. Using numpy arrays, we assigned identification numbers to each value and saved them as "category" and "subcategory" dictionaries, which we then saved as unique DataFrames. These DataFrames connected each category name to a primary key, which we would use later in our database, so they were saved as individual csv files.

The original crowdfunding DataFrame was copied to a campaign_df, where we cleaned up some of the column titles and data types. The campaign_df was merged with both category_df and subcategory_df to include the cat_id and subcat_id for each campaign, and we dropped the columns "category" and "subcategory" since they would be referred to by the foreign keys in their respective id columns in our database. We also dropped unwanted columns "staff_pick" and "spotlight," since we would not use their information for our analyses as both contained Boolean values and we did not have further information regarding the values for either column. The columns were reordered as a cleaned campaign DataFrame, which we saved to a third csv file.

We were given an excel file containing contact information for the organizers of the crowdfunding campaigns in the crowdfunding excel file and tasked with cleaning the information and exporting it as a csv file.

To begin, we loaded the file into a pandas DataFrame and took a look at the data. All of the entries are provided as a JSON-like string formatted as follows:

{“contact_id”: ####, “name”:”John Doe”, “email”:”john.doe@email”}

This formatting did not allow us to create a proper DataFrame from the data and export it in a usable file. For that reason, we had to transform the original data to fit our needs.

First, we took the original data and omitted the first few rows of non-data entries. There were a few lines at the beginning of the excel file that contained notes about the file that were not needed for the final product. As such, these lines were omitted and we put the remainder of the data into a DataFrame.

Second, we iterated through the DataFrame and split each row of data into segments to process the individual items that we needed.

{“contact_id”: ####, “name”:”John Doe”, “email”:”john.doe@email”}

We split each line on the commas to get three individual lines for further processing.

{“contact_id”: ####

“name”:”John Doe”

“email”:”john.doe@email”}

Then, we separated each line on the colon to parse the data even further. At this point, the process was different to extract each piece of information that we needed from the original data.

{“contact_id”	####

For the ‘contact_id’, all that we needed to do was grab the second segment of the parsed information and put it into our list representing the row of clean information.

“Name”   ”John Doe”

For the ‘name’, we needed to grab the second segment of the information but it still required another transformation. We did not need the quotation marks surrounding the name. To fix this, we replaced the quotation marks with nothing and then placed the resulting name into our clean list.

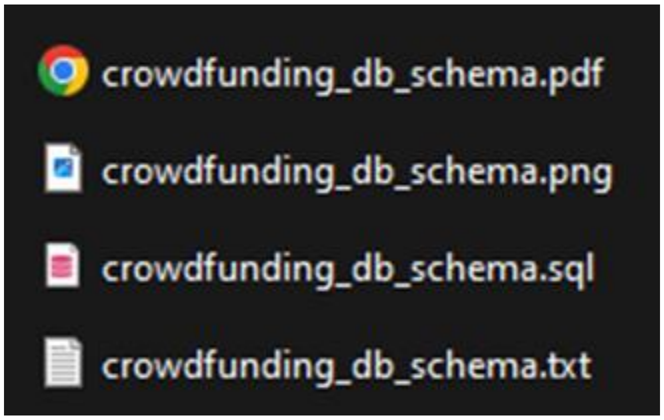
“Email”   ”john.doe@email”}

For the ‘email’, we performed an identical process for the name portion above; however, it required an additional step at the end. At the end of the email, there was an additional curly brace that we needed to omit from the line. After omitting it, we were able to grab the email and add it into our list.

After iterating through all of the DataFrame and adding all of the clean rows of information together, we were able to create another DataFrame with the cleaned data. At that point, all we needed to do was export the data as a csv file to accomplish this task.

Data Loading & Database Creation

For the database schema, we identified all the primary keys and foreign keys within each table. We also identified each field’s datatype. Then we used QuickDBD to create the schema. From there we were able to export four files: a pdf of the schema file, a PNG image file of our entity-relation diagram, a SQL schema file, and a text file of the physical schema.

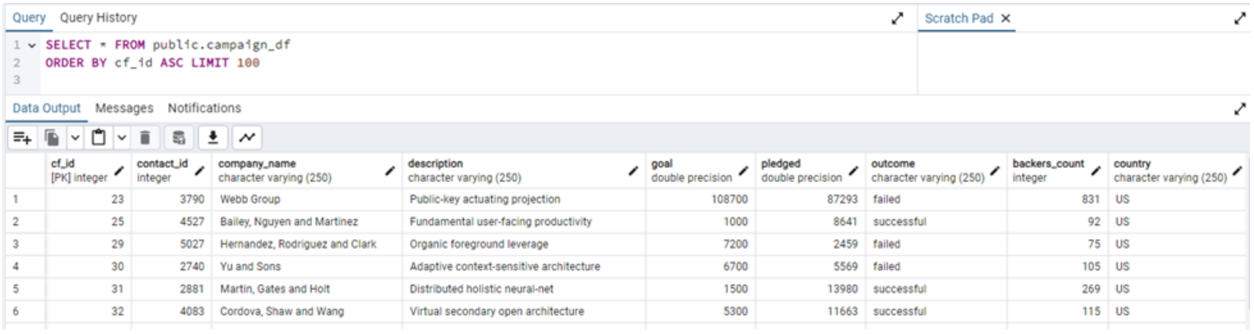


crowdfunding_db_schema.pdf
crowdfunding_db_schema.png
crowdfunding_db_schema.sql
crowdfunding_db_schema.txt

We created a new database named “crowdfunding” within Postgres. With the SQL schema file that we had already created, we were able to create four tables within the database: category_df, subcategory_df, contacts_df, and campaign_df.

After the tables were created, it was time to load the data into the tables. For this process, we used Python code from the upload_csv file.

Below is a screenshot of our SELECT statement and corresponding output to test that the data was loaded into the campaign_df table. We repeated this process for the other 3 tables.



```sql
SELECT * FROM public.campaign_df
ORDER BY cf_id ASC LIMIT 100
```

	cf_id [PK] integer	contact_id integer	company_name character varying (250)	description character varying (250)	goal double precision	pledged double precision	outcome character varying (250)	backers_count integer	country character varying (250)
1	23	3790	Webb Group	Public-key actuating projection	108700	87293	failed	831	US
2	25	4527	Bailey, Nguyen and Martinez	Fundamental user-facing productivity	1000	8641	successful	92	US
3	29	5027	Hernandez, Rodriguez and Clark	Organic foreground leverage	7200	2459	failed	75	US
4	30	2740	Yu and Sons	Adaptive context-sensitive architecture	6700	5569	failed	105	US
5	31	2881	Martin, Gates and Holt	Distributed holistic neural-net	1500	13980	successful	269	US
6	32	4083	Cordova, Shaw and Wang	Virtual secondary open architecture	5300	11663	successful	115	US

Data Analysis

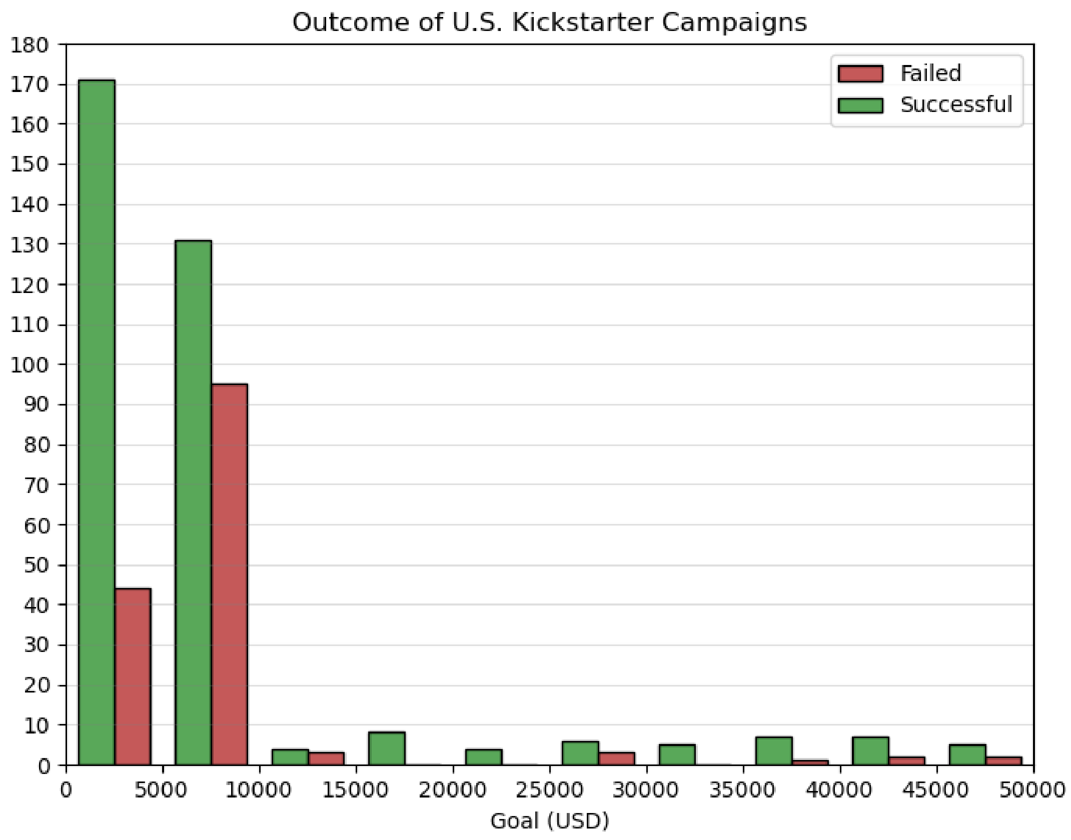
One of the questions that we wanted to answer was “Does the initial goal of a Kickstarter campaign have an effect on its outcome?”

To answer this question, we had to grab the relevant data from our crowdfunding database before we would be able to plot it and potentially find an answer. First, we created a connection to our database using Python’s SQLalchemy library.

After establishing a connection, we ran a quick inspection of the tables in the database to ensure that everything was in order before trying to extract data from the database. With everything in order, we automapped Python classes to match the database tables so that we would be able to execute queries to the database. We established a reference to these classes for each table before continuing.

Then, we started a session and executed the query for our information. This query looked for the outcome, goal, and pledged amount of each campaign in the campaign_df table where the outcome of the campaign was either “successful” or “failed”, the country of the campaign was the United States, and the currency used for the campaign was “USD.” We put all of the extracted data into a Pandas DataFrame.

With all of the information extracted, we could then plot the information. We chose to plot this information as a histogram using the Python library Seaborn. We configured the plot to contain two bars – one for successful campaigns and one for failed campaigns – distributed into 10 bins, each covering a range of $5000, where the bins represent the initial goal amount of the campaign.

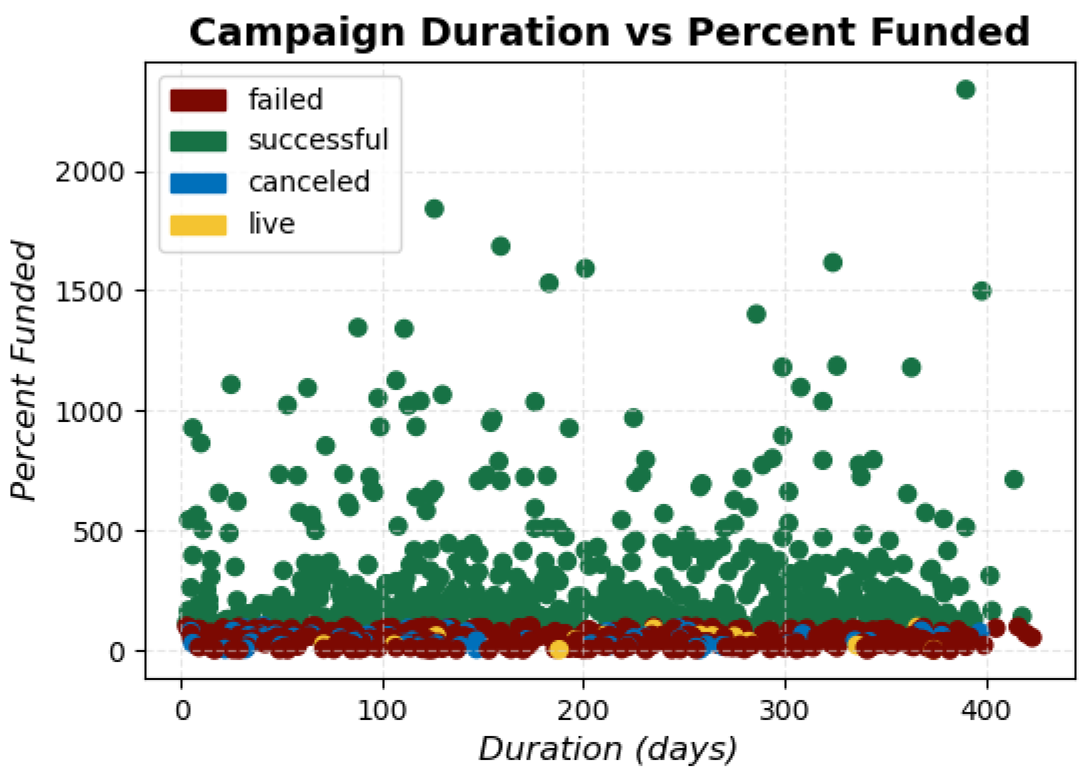


From the graph, there appears to be a slight lull in campaign success beginning at the $5000 to $10,000 range that continues until the $15,000 to $20,000 range. For all other ranges, it seems that the Kickstarter campaigns are often successful given the initial goal amount.

A second question we wanted to analyze was "How does different lengths of run time and receiving different percentages of their goal funding affect each campaign?" To look at this information, we created a second query using raw SQL to aggregate the information we were seeking.

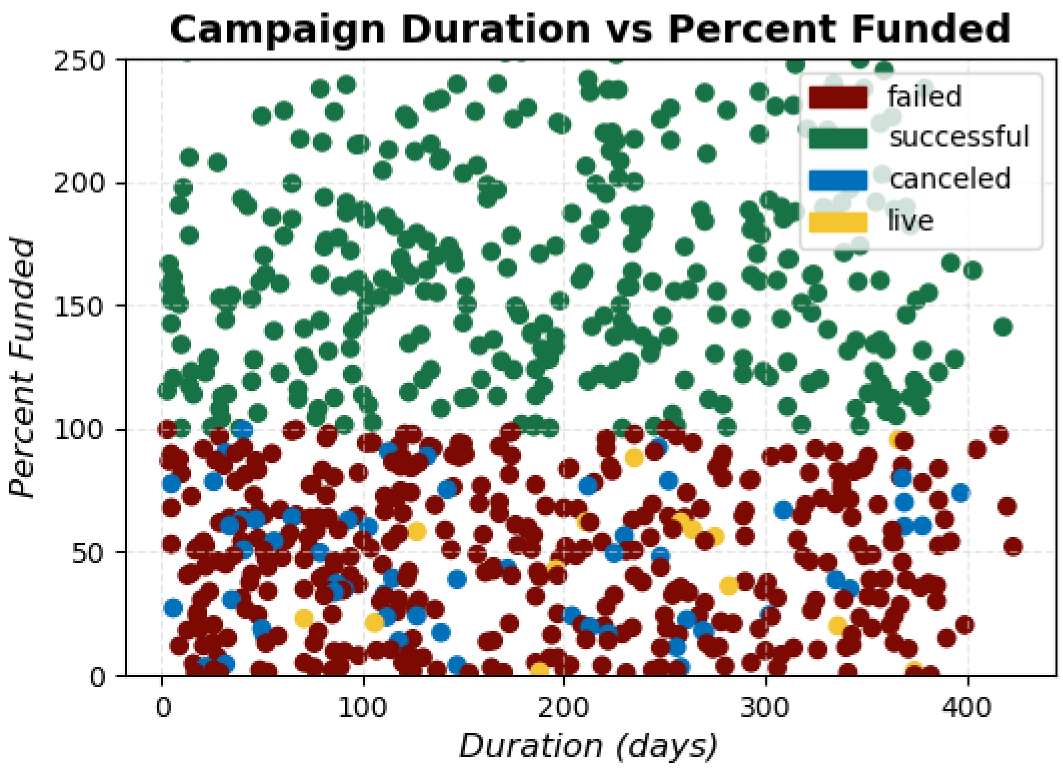
The query selected the "goal", "pledged", and "outcome" columns of the campaign_df, along with extracting the number of days between end_date and launch_date which was saved as "duration," as well as calculating the "percent_funded" column by dividing "pledged" by "goal." The resulting duration_df simplified the information we were looking at to answer our second question.

We created the scatter plot "Campaign Duration vs Percent Funded" which shows durations of 0 to almost 450 days and funding percentages from 0 to almost 2500%. Campaign outcomes (successful, failed, canceled, or live) are depicted by data point colors, as noted in the graph legend.



We can see that all four outcomes are shown all along the duration axis, indicating that duration does not correlate to campaign outcome. We can also see that all campaigns receiving funding over 250% of their goal amount were successful.

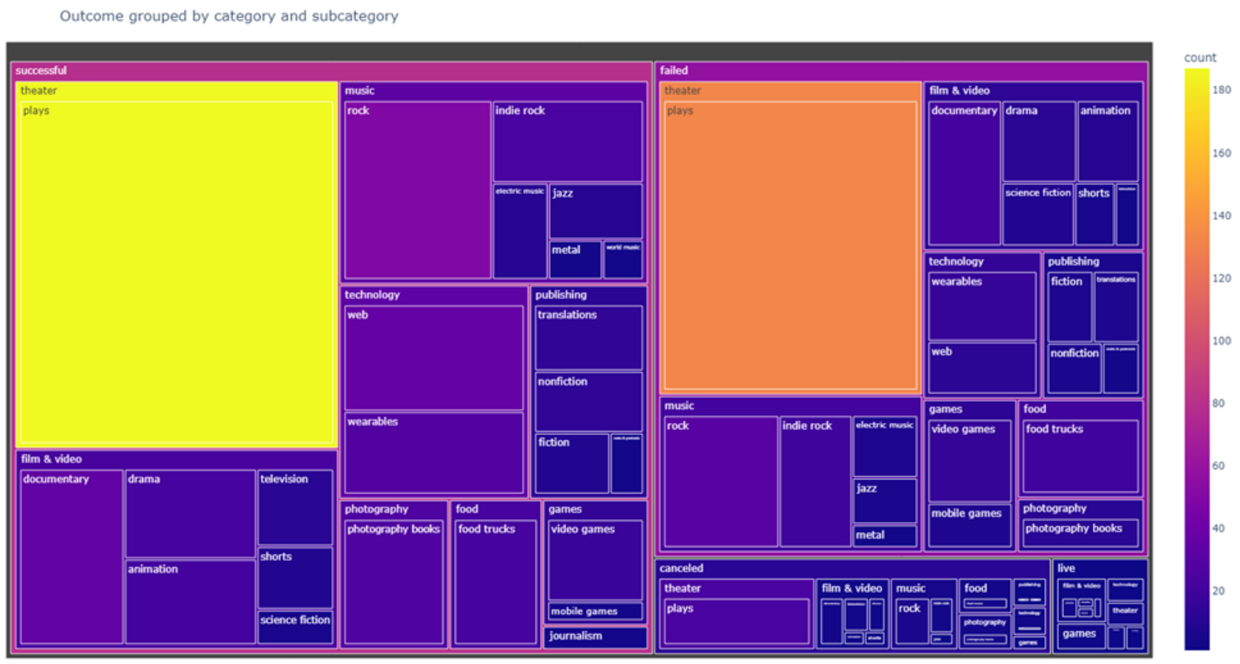
In order to view the information regarding non-successful campaigns more clearly, we recreated this graph but limited the y axis to 250%. This second graph is essentially a zoomed in version of the original “Campaign Duration vs Percent Funded” scatter plot.



With this view, we can see that none of the campaigns were considered successful if they received less than 100% of their goal contributions. However, some of the campaigns that received 100% of their funding were still considered failures or were canceled. We can also see both canceled and live campaigns all throughout the first 100% of funding as well as up to 400 days’ duration, so we would need more information to hypothesize the reasoning for whether or not campaigns were canceled after a certain number of days or after certain percentages of funding.

Based on this graph, we cannot conclude that there is any correlation between campaign duration and percentage of goal funding met. With further research into our source data, we could look into whether these campaigns earned more money initially with popularity or over time with increased marketing, and that information could help us determine what makes these campaigns successful.

To see whether or not the categories or subcategories made a difference in the success of a Kickstarter campaign, we created a treemap that visualized the different outcomes grouped by category and subcategory. We used Plotly Express for the treemap. Currently, the campaign_df only displays category ids and subcategory ids. Instead, we prefer to display the category name and subcategory name, i.e., theater, plays, documentary...etc. In order to achieve this, we had to perform two joins with the category.csv and the subcategory.csv. After the joins were completed, we were able to create a new DataFrame which we used for the treemap. We used Plotly Express for the plot.



Based on this plot, we can see that the category theater and subcategory plays had the highest number of successful campaigns. Concurrently, this combination also had the highest number of failed campaigns. We can conclude that theater plays were the most popular type of campaign to fund, and its chance of success was more than likely.