Crowdfunding Data ETL Workflow

By: Neelam Prasad, Adriana Kuhl, Khalil Locke, Dominique Kelsey

---

Introduction:

For this project, our task was to build an ETL (Extract, Transform, Load) pipeline to analyze data related to a crowdfunding project and its contacts. We were provided with excel files containing data as source, which we then extracted, transformed, and loaded into a PostgreSQL database. The process included using Python and Pandas to process the data. After transforming the data into four separate CSV files, we designed an Entity Relationship Diagram (ERD) and loaded the data into PostgreSQL tables, allowing us to run queries to analyze the information.

PART 1a

Extracting Crowdfunding Data from excel file:

The extraction process was straightforward since the crowdfunding and contacts data were provided as excel files.To extract the data, we first read the crowdfunding excel file into a Pandas DataFrame, which we named `crowdfunding_info_df`. We then generated a summary of the data to understand its structure and contents. Below is a brief view of this extracted data.

Table 1: Summary of Crowdfunding Data (crowdfunding_info_df)

```
In [119…
# Get a brief summary of the crowdfunding_info DataFrame.
crowdfunding_info_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 15 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   cf_id                  1000 non-null   int64
 1   contact_id             1000 non-null   int64
 2   company_name           1000 non-null   object
 3   blurb                  1000 non-null   object
 4   goal                   1000 non-null   int64
 5   pledged                1000 non-null   int64
 6   outcome                1000 non-null   object
 7   backers_count          1000 non-null   int64
 8   country                1000 non-null   object
 9   currency               1000 non-null   object
 10  launched_at            1000 non-null   int64
 11  deadline               1000 non-null   int64
 12  staff_pick             1000 non-null   bool
 13  spotlight              1000 non-null   bool
 14  category & sub-category  1000 non-null   object
dtypes: bool(2), int64(7), object(6)
memory usage: 103.6+ KB
```

Transformation:

The transformation phase involved processing and organizing the data so that it would be usable in PostgreSQL for analysis. This stage included creating four CSV files from the original excel files i.e. crowdfunding.xlsx and contacts.xlsx: `Campaign`, `Category`, `Subcategory`, and `Contacts`.

Category and Subcategory DataFrames:

Initially, the `crowdfunding_info_df` included a column that combined both category and subcategory information. We separated these into two distinct columns using the `split()` function in Pandas, as shown in **Table 2**.

Table 2: Splitting the Category and Subcategory Column into Two Separate Columns

```python
# Assign the category and subcategory values to category and subcategory columns.
crowdfunding_info_df[['category','subcategory']] = crowdfunding_info_df['category & sub-category'].str.split("/", expand =True)
crowdfunding_info_df
```

	cf_id	contact_id	company_name	blurb	goal	pledged	outcome	backers_count	country	currency	launched_at	deadline	staff_pick	spotlight	category & sub-category	category	subcategory
0	147	4661	Baldwin, Riley and Jackson	Pre-emptive tertiary standardization	100	0	failed	0	CA	CAD	1581573600	1614578400	False	False	food/food trucks	food	food trucks
1	1621	3765	Odom Inc	Managed bottom-line architecture	1400	14560	successful	158	US	USD	1611554400	1621918800	False	True	music/rock	music	rock
2	1812	4187	Melton, Robinson and Fritz	Function-based leadingedge pricing structure	108400	142523	successful	1425	AU	AUD	1608184800	1640844000	False	False	technology/web	technology	web
3	2156	4941	Mcdonald, Gonzalez and Ross	Vision-oriented fresh-thinking conglomeration	4200	2477	failed	24	US	USD	1634792400	1642399200	False	False	music/rock	music	rock
4	1365	2199	Larson-Little	Proactive foreground core	7600	5265	failed	53	US	USD	1608530400	1629694800	False	False	theater/plays	theater	plays
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
995	2986	3684	Manning-Hamilton	Vision-oriented scalable definition	97300	153216	successful	2043	US	USD	1609221600	1622350800	False	True	food/food trucks	food	food trucks
996	2031	5784	Butler LLC	Future-proofed upward-trending migration	6600	4814	failed	112	US	USD	1634274000	1638252000	False	False	theater/plays	theater	plays
997	1627	1498	Ball LLC	Right-sized full-range throughput	7600	4603	canceled	139	IT	EUR	1636174800	1639116000	False	False	theater/plays	theater	plays
998	2175	6073	Taylor, Santiago and Flores	Polarized composite customer loyalty	66600	37823	failed	374	US	USD	1602133200	1618117200	False	True	music/indie rock	music	indie rock
999	1788	4939	Hernandez, Norton and Kelley	Expanded eco-centric policy	111100	62819	canceled	1122	US	USD	1609308000	1629262800	False	False	food/food trucks	food	food trucks

1000 rows × 17 columns

Next, we created unique lists for both the category and subcategory columns and grouped them by unique characteristics. Using list comprehensions, we prefixed the values with "cat" for category IDs and "subcat" for subcategory IDs. This allowed us to create a unique identifier for each value.

Image 1: Code Reference for List Comprehension in Data Transformation

In [125…

```python
# Use a list comprehension to add "cat" to each category_id.

cat_ids = [f"cat{cat_id}" for cat_id in category_id]
print(cat_ids)

# Use a list comprehension to add "subcat" to each subcategory_id.
scat_ids = [f"subcat{subcat_id}" for subcat_id in subcategory_id]
print(scat_ids)
```

```
['cat1', 'cat2', 'cat3', 'cat4', 'cat5', 'cat6', 'cat7', 'cat8', 'cat9']
['subcat1', 'subcat2', 'subcat3', 'subcat4', 'subcat5', 'subcat6', 'subcat7', 'subcat8', 'subcat9', 'subcat10', 'subcat11',
'subcat12', 'subcat13', 'subcat14', 'subcat15', 'subcat16', 'subcat17', 'subcat18', 'subcat19', 'subcat20', 'subcat21', 'subc
at22', 'subcat23', 'subcat24']
```

After this, we created new DataFrames for the `category` and `subcategory` columns, ensuring that the values were properly ordered and unique. These transformed DataFrames were then exported as CSV files for loading into PostgreSQL.

Table 3: Final Category and Subcategory DataFrames

In [127…

```
category_df
```

Out[127…

	category_id	category
0	cat1	food
1	cat2	music
2	cat3	technology
3	cat4	theater
4	cat5	film & video
5	cat6	publishing
6	cat7	games
7	cat8	photography
8	cat9	journalism

In [128…

```
subcategory_df
```

Out[128…

	subcategory_id	subcategory
0	subcat1	food trucks
1	subcat2	rock
2	subcat3	web
3	subcat4	plays
4	subcat5	documentary
5	subcat6	electric music
6	subcat7	drama
7	subcat8	indie rock
8	subcat9	wearables
9	subcat10	nonfiction

Campaign DataFrame:

For the `Campaign` DataFrame, we renamed column headers and converted the `Goal` and `Pledged` columns from integer types to float. We also adjusted the timestamp fields to use a proper datetime format instead of UTC timestamps. After merging the transformed `Category` and `Subcategory` data with the campaign data, we obtained the final `campaign_revised_df`, as shown in **Table 4**.

Table 4: Final Campaign DataFrame (campaign_revised_df)

Out[136…

description	goal	pledged	outcome	backers_count	country	currency	launch_date	end_date	category_id	subcategory_id
Seamless 4thgeneration methodology	84600.0	134845.0	successful	1249	US	USD	2020-04-12 05:00:00+00:00	2021-03-24 05:00:00+00:00	cat5	subcat11
Down-sized analyzing challenge	9000.0	14455.0	successful	129	US	USD	2020-10-28 05:00:00+00:00	2021-02-15 06:00:00+00:00	cat5	subcat11
Seamless coherent parallelism	600.0	4022.0	successful	54	US	USD	2021-07-27 05:00:00+00:00	2021-11-30 06:00:00+00:00	cat5	subcat11
Pre-emptive impactful model	9500.0	4460.0	failed	56	US	USD	2021-06-17 05:00:00+00:00	2021-08-27 05:00:00+00:00	cat5	subcat11
Stand-alone mobile customer loyalty	41700.0	138497.0	successful	1539	US	USD	2020-08-30 05:00:00+00:00	2021-08-04 05:00:00+00:00	cat5	subcat11
...	...	...	...	...	...	...	...	...	...	...
Multi-tiered systematic knowledge user	42700.0	97524.0	successful	1681	US	USD	2020-12-18 06:00:00+00:00	2021-03-15 05:00:00+00:00	cat3	subcat3

PART 1b

Creating the contacts.csv File from Excel

To create the `contacts.csv` file from the provided Excel file, we followed a series of steps in Python using the Pandas library. Below is a detailed breakdown of the process:

Reading the Excel File: We started by reading the `contacts.xlsx` file into a Pandas DataFrame, skipping the first two rows as they were headers and not actual data.

```python
# Remove the first row
contact_info_df = contact_info_df.drop(0).reset_index(drop=True)

# Check the DataFrame
contact_info_df.columns = ['contact_info']
contact_info_df.head()
```

	contact_info
0	{"contact_id": 4661, "name": "Cecilia Velasco", "email": "cecilia.velasco@rodrigues.fr"}
1	{"contact_id": 3765, "name": "Mariana Ellis", "email": "mariana.ellis@rossi.org"}
2	{"contact_id": 4187, "name": "Sofie Woods", "email": "sofie.woods@riviere.com"}
3	{"contact_id": 4941, "name": "Jeanette Iannotti", "email": "jeanette.iannotti@yahoo.com"}
4	{"contact_id": 2199, "name": "Samuel Sorgatz", "email": "samuel.sorgatz@gmail.com"}

Extracting Data from the 'contact_info' Column: The `contact_info` column in the dataset contained JSON strings that we needed to convert into dictionaries. We iterated over each row, parsed the JSON string in the `contact_info` column, and appended the resulting dictionaries to a list.

```python
    contact_dict = json.loads(row['contact_info'])

    # Append the dictionary to the list
    dict_values.append(contact_dict)

# Print out the list of values for each row.
print(dict_values)
```


```
[{'contact_id': 4661, 'name': 'Cecilia Velasco', 'email': 'cecilia.velasco@rodrigues.fr'}, {'contact_id': 3765, 'name': 'Mariana El
lis', 'email': 'mariana.ellis@rossi.org'}, {'contact_id': 4187, 'name': 'Sofie Woods', 'email': 'sofie.woods@riviere.com'}, {'conta
ct_id': 4941, 'name': 'Jeanette Iannotti', 'email': 'jeanette.iannotti@yahoo.com'}, {'contact_id': 2199, 'name': 'Samuel Sorgatz',
'email': 'samuel.sorgatz@gmail.com'}, {'contact_id': 5650, 'name': 'Socorro Luna', 'email': 'socorro.luna@hotmail.com'}, {'contact_
id': 5889, 'name': 'Carolina Murray', 'email': 'carolina.murray@knight.com'}, {'contact_id': 4842, 'name': 'Kayla Moon', 'email':
'kayla.moon@yahoo.de'}, {'contact_id': 3280, 'name': 'Ariadna Geisel', 'email': 'ariadna.geisel@rangel.com'}, {'contact_id': 5468,
'name': 'Danielle Ladeck', 'email': 'danielle.ladeck@scalfaro.net'}, {'contact_id': 3064, 'name': 'Tatiana Thompson', 'email': 'tat
iana.thompson@hunt.net'}, {'contact_id': 4904, 'name': 'Caleb Benavides', 'email': 'caleb.benavides@rubio.com'}, {'contact_id': 129
```


Creating the `contact_df` DataFrame: We then created a new DataFrame, `contact_df`, from the list of dictionaries (`dict_values`), which contained the detailed information for each contact.

```
contact_df = pd.DataFrame(dict_values)
contact_df
```

	contact_id	name	email
0	4661	Cecilia Velasco	cecilia.velasco@rodrigues.fr
1	3765	Mariana Ellis	mariana.ellis@rossi.org
2	4187	Sofie Woods	sofie.woods@riviere.com
3	4941	Jeanette Iannotti	jeanette.iannotti@yahoo.com
4	2199	Samuel Sorgatz	samuel.sorgatz@gmail.com

Splitting the 'name' Column: The `name` column in `contact_df` contained full names (e.g., "John Doe"), which we split into two separate columns: `first_name` and `last_name`.

Dropping the Original 'name' Column: After splitting the names, we dropped the original `name` column, as it was no longer needed.

Selecting Relevant Columns: We selected the necessary columns (`contact_id`, `first_name`, `last_name`, and `email`) to clean up the DataFrame.

```
contacts_df_clean = contact_df[['contact_id', 'first_name', 'last_name', 'email']]
contacts_df_clean
```

	contact_id	first_name	last_name	email
0	4661	Cecilia	Velasco	cecilia.velasco@rodrigues.fr
1	3765	Mariana	Ellis	mariana.ellis@rossi.org
2	4187	Sofie	Woods	sofie.woods@riviere.com
3	4941	Jeanette	Iannotti	jeanette.iannotti@yahoo.com
4	2199	Samuel	Sorgatz	samuel.sorgatz@gmail.com

Exporting the DataFrame as a CSV: Finally, we exported the cleaned `contacts_df_clean` DataFrame as a CSV file, which we named `contacts.csv`. This file would later be used for loading into the PostgreSQL database.

Loading the Data into PostgreSQL Database

After transforming the data into the required CSV format, the next step was to load the data into our PostgreSQL database. This part of the project displays data modeling, data engineering, and data analysis by using Structured Query Language (SQL). Applying our knowledge of DataFrames and tabular data, we created entity relationship diagrams (ERDs), imported data into a database, troubleshooted common errors, and created queries that use data to answer questions. We used Python's Pandas library in combination with SQLAlchemy to achieve this. Below is the step-by-step process:

1.Establishing the Database Connection: First, we established a connection to our PostgreSQL database using the SQLAlchemy engine. We provided the necessary credentials such as the username, password, host, and database name.

2.Inspecting the Database: To ensure that the connection was successfully established, we used SQLAlchemy's inspector to retrieve and print out the names of the tables within the database. Additionally, we printed the columns and their data types for each table to verify the schema.

```
# CONNECT TO POSTGRES
USERNAME = "postgres"
PASSWORD = "password"
HOST = "localhost"
PORT = 5432
DATABASE = "crowdfunding_db"
connection_str = f"postgresql://{USERNAME}:{PASSWORD}@{HOST}:{PORT}/{DATABASE}"

# Create Engine
engine = create_engine(connection_str)
```

```
# Create the inspector and connect it to the engine
inspector = inspect(engine)

# Collect the names of tables within the database
tables = inspector.get_table_names()

# Using the inspector to print the column names within the 'dow' table and its types
for table in tables:
    print(table)
    print("--------")
    columns = inspector.get_columns(table)
    for column in columns:
        print(column["name"], column["type"])

    print()
```

3.Loading Data into the Database: Once the database connection was confirmed, we loaded each CSV file into the appropriate table in PostgreSQL. We used Pandas' `to_sql()` method, specifying the table names and connection engine. The `if_exists="append"` parameter ensured that the data was appended to the existing table without overwriting it. The `method="multi"` parameter was used to optimize the insertion by inserting multiple rows at once.

The data was loaded in the following sequence:

Contacts Table:

```
contacts_df = pd.read_csv("Resources/contacts.csv")

# Write to SQL (USING con=engine)
contacts_df.to_sql(name="contacts", con=engine, index=False, if_exists="append", method="multi")
```

```
1000
```

Category Table:

```
category_df = pd.read_csv("Resources/category.csv")

# Write to SQL (USING con=engine)
category_df.to_sql(name="category", con=engine, index=False, if_exists="append", method="multi")
```

```
9
```

Subcategory Table:

```
subcategory_df = pd.read_csv("Resources/subcategory.csv")

# Write to SQL (USING con=engine)
subcategory_df.to_sql(name="subcategory", con=engine, index=False, if_exists="append", method="multi")
```

```
24
```

Campaign Table:

```
campaign_df = pd.read_csv("Resources/campaign.csv")

# Write to SQL (USING con=engine)
campaign_df.to_sql(name="campaign", con=engine, index=False, if_exists="append", method="multi")
```

```
1000
```

After loading all the data, we confirmed that each table contained the correct number of records by querying the database or checking the table contents via a database management tool (e.g., PgAdmin).

By following this approach, we successfully loaded the transformed CSV data into the PostgreSQL database, making it available for further analysis and querying.

Raw SQL vs. ORM Syntax: Pros and Cons

While we used raw SQL and Pandas to load the data into PostgreSQL in this project, an alternative approach is to use an ORM (Object-Relational Mapping) library like SQLAlchemy's ORM syntax. ORM provides a higher-level abstraction, allowing you to interact with the database using Python **classes** and **objects** rather than writing explicit SQL queries.

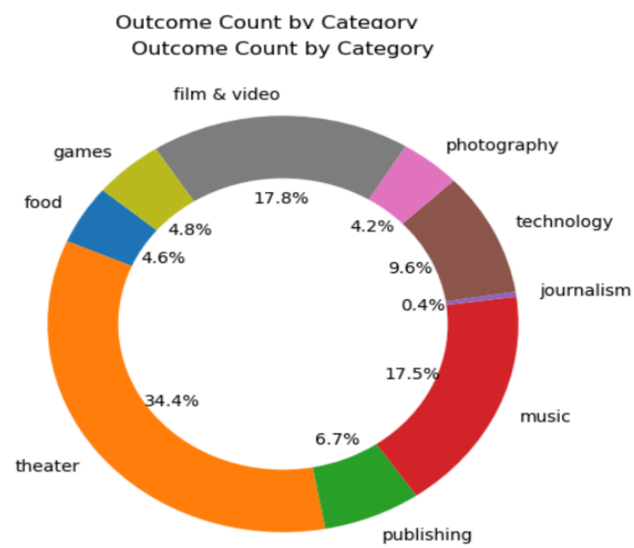
- **Pros of ORM:**
    - **Ease to use:** ORM allows developers to interact with the database using Python objects, reducing the need to write raw SQL.
    - **Database Abstraction:** ORM provides database abstraction, allowing us to switch between different database systems with minimal changes to the code.
- **Cons of ORM:**
    - **Less efficient** than hand-written raw SQL queries for large-scale or highly complex operations.
    - **Less Control:** It may not provide the same level of fine-grained control over complex queries, transactions, or optimizations that raw SQL can offer.

In our project, we chose to use raw SQL for its simplicity and direct control over database operations, but for larger applications or projects requiring complex database interactions, ORM could be a valuable alternative.

Sample Leaderboards and graphs to show our analysis:

Campaigns distributed across different categories based on their outcome count

	category	outcome_count
0	food	46
1	theater	344
2	publishing	67
3	music	175
4	journalism	4
5	technology	96
6	photography	42
7	film & video	178
8	games	48



Top 10 campaigns who raised the highest amount of money compared to their goal

	company_name	goal	pledged	percentage_funded
0	Williams-Jones	600.0	14033.0	23.39
1	Garza-Bryant	800.0	14725.0	18.41
2	Smith, Love and Smith	800.0	13474.0	16.84
3	Ramirez-Myers	900.0	14547.0	16.16
4	Green-Carr	900.0	14324.0	15.92
5	Smith-Schmidt	900.0	13772.0	15.30
6	Porter-George	1000.0	14973.0	14.97
7	Petersen and Sons	900.0	12607.0	14.01
8	Wong-Walker	900.0	12102.0	13.45
9	Turner-Davis	600.0	8038.0	13.40

Contact Details of top 5 highest performing campaigns

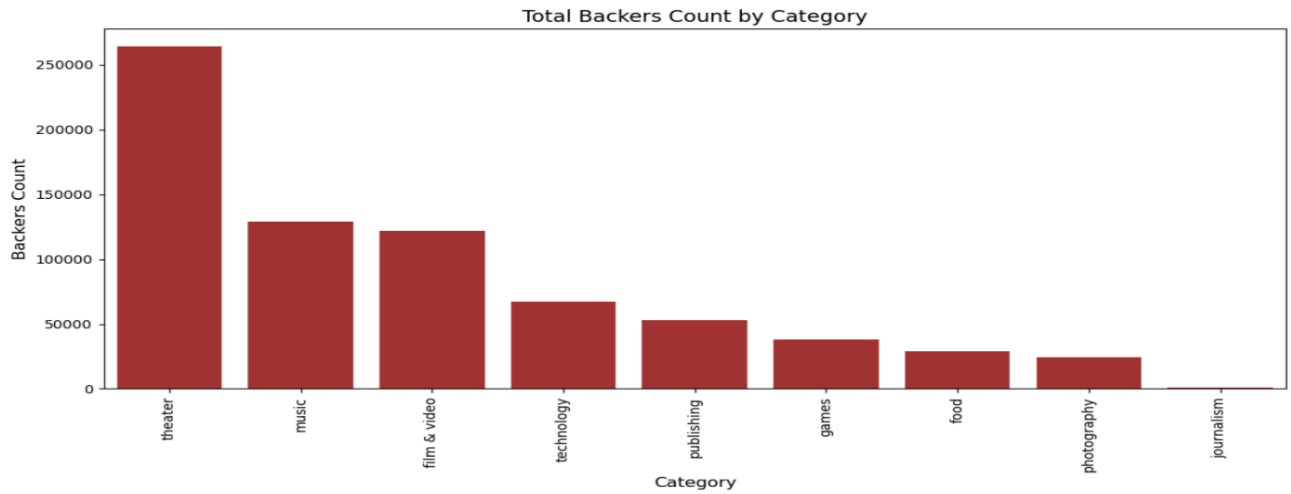
```sql
--1.What are the contact details for the top 5 highest-performing campaigns?
SELECT
    c.company_name,
    c.pledged,
    con.first_name,
    con.last_name,
    con.email
FROM
    campaign c
JOIN
    contacts con ON c.contact_id = con.contact_id
ORDER BY
    c.pledged DESC
LIMIT 5;
```

ta Output Messages Notifications

company_name character varying (50)	pledged numeric (10,2)	first_name character varying (25)	last_name character varying (25)	email character varying (50)
Jackson Inc	199110.00	Casey	Flores	casey.flores@baggio.org
Jordan-Acosta	198628.00	Ludovica	Arellano	ludovica.arellano@morandi-argento.com
Perez Group	197728.00	Severino	Linares	severino.linares@angeli.com
Smith-Wallace	197024.00	Cornelio	Guardado	cornelio.guardado@gmail.com
Hicks, Wall and Webb	197018.00	Roberto	Guyot	roberto.guyot@bennett.com

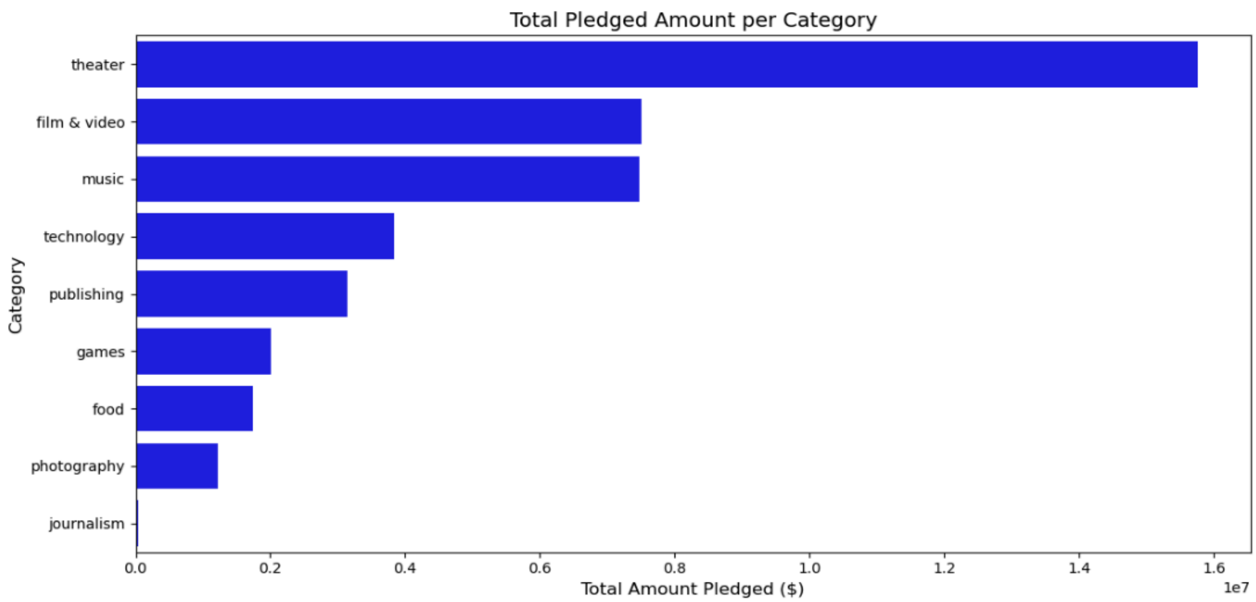
Categories with total number of backers in descending order

	category	total_backers
0	theater	264269
1	music	129002
2	film & video	121875
3	technology	67494
4	publishing	52619
5	games	37662
6	food	28846
7	photography	24044
8	journalism	1194



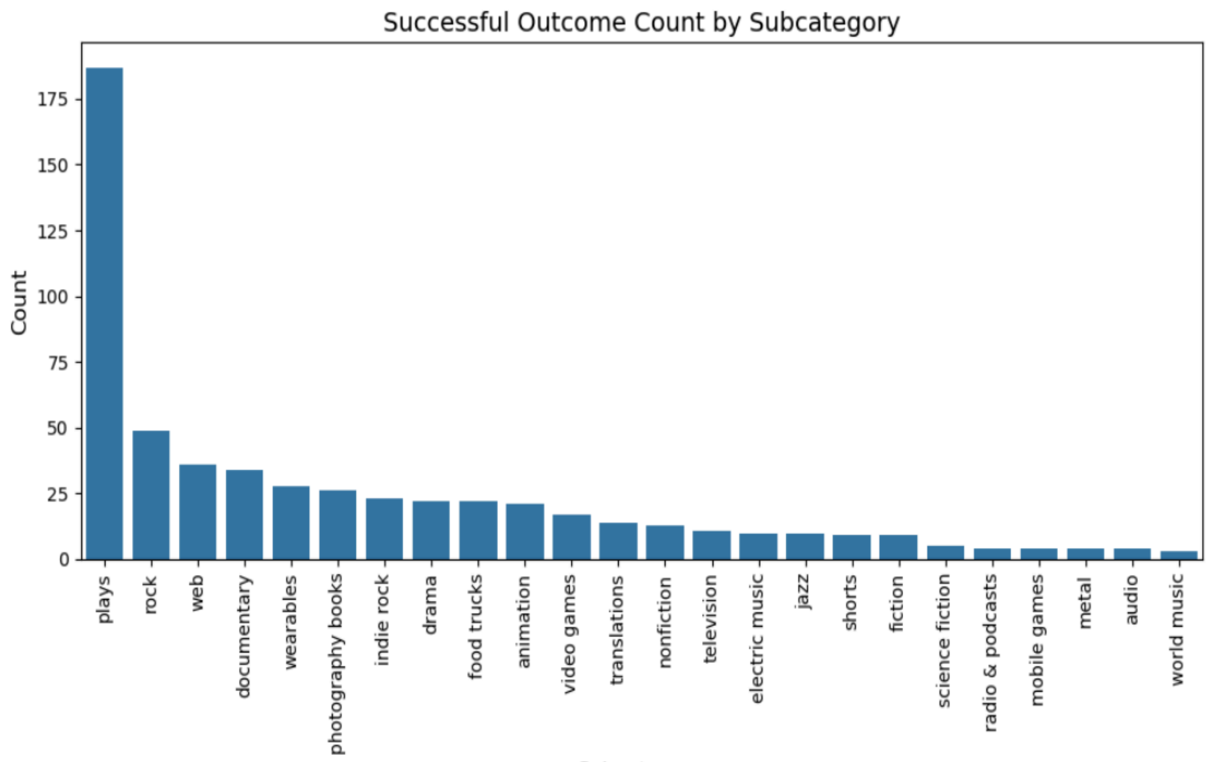
The amount of money pledged in total across different crowdfunding categories

	category	total_pledged
0	theater	15763227.0
1	film & video	7510076.0
2	music	7480097.0
3	technology	3833725.0
4	publishing	3149827.0
5	games	2015817.0
6	food	1735179.0
7	photography	1223931.0
8	journalism	36176.0



Leaderboard showing count of 'successful' outcomes for each subcategory, grouped by the count of successful outcomes in descending order

	subcategory	successful_count
0	plays	187
1	rock	49
2	web	36
3	documentary	34
4	wearables	28
5	photography books	26
6	indie rock	23
7	drama	22
8	food trucks	22
9	animation	21



Conclusion

The ETL process in the crowdfunding project helped us clean, organize, and transform the raw data into a format that could be easily analyzed. By extracting data from CSV files, transforming it into structured tables, and loading it into a PostgreSQL database, we were able to gain valuable insights into the crowdfunding industry. Using tools like Python's Pandas and SQLAlchemy, along with PostgreSQL for managing the data, we set up a solid foundation for future analysis. This will allow for better decision-making and more effective strategies in future crowdfunding campaigns.