Movie_ETL Analysis

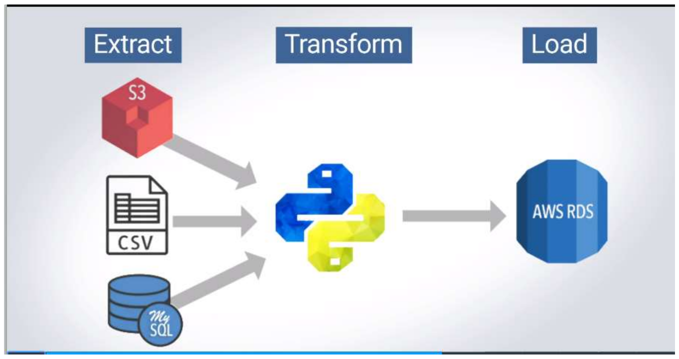
Description

Amazing Prime is an online streaming movie service. To improve profit margins, they are wanting to develop an algorithm which will predict which low budget movies released will become popular among their customers – thereby enabling them to purchase the streaming rights at a low cost – before the movie becomes popular.

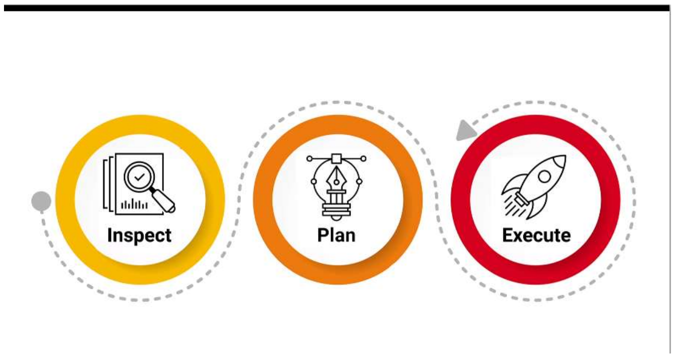
In order to facilitate creation of the algorithm, they have decided to hold a hackathon and see if they can source the algorithm from the data science community. In preparation for the hackathon they have pulled movie and rating data from 2 different sources:

- Wikipedia scrape for movies from 1990
- Rating Data from the MovieLens website

In this module, we used the Extract, Transform, Load (ETL) process to take 3 datafiles ('wikipedia.movies.json', 'movies_metadata.csv', 'ratings.csv') and prepare them for Amazing Prime's hackathon. The general ETL process is shown below. We stored our cleaned data in an SQL database called 'movie_data' and within 2 tables called 'movies' and 'ratings'.



Data cleaning uses an iterative process in which we Inspect, Plan, and Execute multiple times on the data before it is clean enough to be used or we reach a point of diminishing returns where further cleaning is too time-consuming for the incremental results.



Other Resources

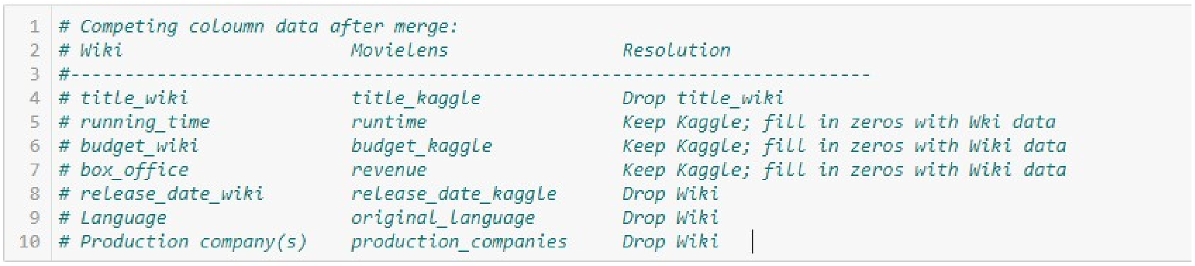
- Python 3.6 (Python 3 environment)
- Python Libraries – json, pandas, re, numpy, sqlalchemy, psycopg2, config, time.
- PostgresQL 11
- Data files as outlined above
- GitHub Repository – Movies_ETL

Pseudo Code (High Level)

The final script stored in the repository is Challenge8_ETL. This file uses about 90% of the code from our Module 8 exercises (named etl_file) – also in the repository -- but with exploratory code removed and comments added resulting in a clean script. Below is Pseudo code providing guidance to the python script:

- Import dependencies and config file containing database password
- Define path to data source files and read them into a dictionary and dataframes.

- Wiki_movies data (7311 rows) – defined clean_movie function to:
    - Remove rows with Null data for Director, Directed by, and imdb_link
    - Removed rows containing data in No. of episodes indicating not a movie
    - Checked alternative titles (different languages) and created new column with Alt_titles.
    - Changed column names to eliminate obvious duplicates (i.e. 'Music by' to 'Composer')
    - Using clean-movie function output, create a wiki_movies_df.
    - Delete columns in wiki_movies_df with less than 90% of rows filled with data.
    - Clean up data types, formats, strings, etc in:
        - Box_office, Budget, Release Date, Running Time, Id, Popularity.
- Ratings data (26 million plus rows)
    - Set timestamp
- Merged Data from wiki_movies_df and kaggle_meta_df
- Cleaning was then performed on the merged data 'movies_df' as follows:



```
# Competing coloumn data after merge:
# Wiki                      Movielens                 Resolution
#--------------------------------------------------------------------------
# title_wiki                title_kaggle              Drop title_wiki
# running_time              runtime                   Keep Kaggle; fill in zeros with Wki data
# budget_wiki               budget_kaggle             Keep Kaggle; fill in zeros with Wiki data
# box_office                revenue                   Keep Kaggle; fill in zeros with Wiki data
# release_date_wiki         release_date_kaggle       Drop Wiki
# Language                  original_language         Drop Wiki
# Production company(s)     production_companies      Drop Wiki
```

    - Reorder columns into logical groups
    - Rename column names
- Rating Data was was grouped by 'movieId' and 'rating', then counted creating new data for statistical analysis. That was then merged with the movies_df data.

PostgresQL Data Loads

2 separate data loads are performed in the script:

- movies_df.to_sql - loading data into the 'movies' table in the SQL database
- ratings.csv - loading data into the 'ratings' table in the SQL database.

In order to facilitate an on-going datapipeline, consideration was given to either "replace" or "append" the data. Since each new pull of the data from its origins would have duplicate records from the previous load, it was decided to use the "replace" method which deletes and replaces all previous data.

This also prevents the dataset from growing abnormally large that would happen with "append."

The "if_exists = replace" statement is an attribute of the data.to_sql statement.

Creating an Ongoing Pipeline – 5 Assumptions

1) The data structures (JSON and CSV) and sources (Wikipedia and Kaggle) will continue to be available without change.
    a. If major changes were to take place in the formatting, then read/load scripts would need to be modified accordingly.
2) Column Names are stable
    a. Much of the cleaning was done to delete columns without data or cleanup duplicate or near duplicate column names. Try-Else statements can be added to ensure code continues if errors are encountered.
3) Low-Budget Films Versus Quantity of Rating Data
    a. With a total of 27 million records of ratings data, it was assumed that enough data exists (statistically) on the low budget films to extract a sample, develop the algorithm then test predictions. Filtering the rating data by "Budget" could be a further tets to determine the quantity of ratings available.
4) Matches Between Ratings Data and Movie Data
    a. It is assumed that there is suffient matches/keys to enable joining the ratings data with the low-budget files. Besides the shear number of ratings, being able to match that data to its corresponding film from data pulled from a different source is of key importance.

5) Correlation of Ratings Versus Movie Data
    a. It is assumed that there is sufficient movie attributes data for the participants of the hackathon to find meaningful correlations on the low-budget films upon which to build their algorithm. Without sufficient correlations (i.e. a key missing attribute that determines popularity of streaming) a predictive algorithm can not be built for Prime.