CA270: Build and Analyse a Data Mining Algorithm

Humayun Awan and Luqman Rostam

S1 - Research Question:

Finding a suitable and appropriate research question was a major and extremely important part of this project, we approached it by finding the dataset first and proposing research questions after. In the end our research question was “Predicting the Success of a Movie”, using revenue as our measure of success.

We spent quite a while on this part of the project. This was due to many factors, mainly being that we had another idea we were hoping to pursue which was in talks, as we liked the idea of using primary data we had collected first-hand. Unfortunately the idea fell through as we required permissions from various different organisations and we could not promise to provide results which were of an industry standard as this was our first time using these technologies.

When setting out to find a new idea we focused on three main points:

- Do we have enough data available?
- Will we be able to complete it within our time frame and current level of expertise?
- Is it interesting?

The first two questions are rather self-explanatory, but the third point of whether this new idea would be interesting was especially important to us. If we were to pick up a simple and straightforward dataset which neither of us found interesting, we would lack motivation in doing the project to the best of our abilities.

Fortunately, we were able to find a topic which interested both of us which was Movies/Cinema. The research question itself also took quite some time to finalise as it would largely depend on the dataset we chose and what we could interpret from the data. This made choosing our dataset a pivotal step, as it could be the difference between an accurate machine learning algorithm and a very poor inaccurate algorithm.

Upon our first attempt at the project we tried multiple different datasets, however these datasets either lacked the necessary features/columns or the necessary size. We eventually found a suitable dataset which consisted of 6 different csv files, the dataset itself had plenty of features, both categorical and numerical data. On top of all this, the dataset consisted of almost 50,000 lines

S1 - Dataset Description:

[Movies Dataset]

To load and prepare our data we used the pandas library. The pandas library is tailored to use on datasets, it has many useful functions for e.g manipulating data, dropping columns, dropping null values, merging datasets, adding features etc.

```
import pandas as pd
```

Upon downloading a zip file, we found that our dataset was more in depth than we imagined. We had multiple csv files full of interesting data, however the majority of data that we used was located in the “movies_metadata” dataset. This file in itself was massive and contained 24 columns and 45,466 rows of data.

```
movies_meta_data.shape
```

```
(45466, 24)
```

Shape of our data (rows, columns)

Large dataset

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45466 entries, 0 to 45465
Data columns (total 24 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   adult                  45466 non-null  object
 1   belongs_to_collection  4494 non-null   object
 2   budget                 45466 non-null  object
 3   genres                 45466 non-null  object
 4   homepage               7782 non-null   object
 5   id                     45466 non-null  object
 6   imdb_id                45449 non-null  object
 7   original_language      45455 non-null  object
 8   original_title         45466 non-null  object
 9   overview               44512 non-null  object
 10  popularity             45461 non-null  object
 11  poster_path            45080 non-null  object
 12  production_companies   45463 non-null  object
 13  production_countries   45463 non-null  object
 14  release_date           45379 non-null  object
 15  revenue                45460 non-null  float64
 16  runtime                45203 non-null  float64
 17  spoken_languages       45460 non-null  object
 18  status                 45379 non-null  object
 19  tagline                20412 non-null  object
 20  title                  45460 non-null  object
 21  video                  45460 non-null  object
 22  vote_average           45460 non-null  float64
 23  vote_count             45460 non-null  float64
dtypes: float64(4), object(20)
memory usage: 8.3+ MB
```

When preparing the data we realised that some of the columns in the csv files were frankly useless for our machine learning algorithm.

Some of these include: homepage, imdb_id, and poster_path, adult, video.

We ended up dropping as many of these redundant columns as possible and in our finished dataset we had 12 columns.

We also ran into many problems when it came to these columns, which we explain in detail in our data preparation process.

S1 - Data Preparation:

As mentioned earlier, some of the information in the dataset was redundant and added to the confusion when cleaning our dataset and building our model. After fixing the problems we identified and dropped redundant columns and reindexed the remaining features to fit our plans.

```
combined_df = combined_df.drop(columns=["genres", "production_companies",
                        "production_countries", "release_date"])

combined_df = combined_df.reindex(columns=["id", "belongs_to_collection", "title", "release_year",
                            "genres_id", "genres_name", "budget", "revenue", "original_language",
                            "production_companies_id", "production_companies_name",
                            "production_countries_id", "production_countries_name",
                            "runtime", "vote_average", "vote_count", "popularity", "crew"])

print("DROPPED AND REINDEXED")
```

After dropping the redundant columns the final version of our dataset looked like this.

These were some of the issues we consistently ran into

- Columns containing JSON objects.
- Columns in incorrect data types
- Multiple-value rows
- NULL values – for important columns

```
<class 'pandas.core.frame.DataFrame'>
Index: 5392 entries, 0 to 5391
Data columns (total 12 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   id                  5392 non-null   object
 1   title               5392 non-null   object
 2   release_year        5392 non-null   object
 3   genre               5392 non-null   object
 4   production_company  5392 non-null   object
 5   production_country  5392 non-null   object
 6   original_language   5392 non-null   object
 7   budget              5392 non-null   int64
 8   runtime             5392 non-null   float64
 9   vote_average        5392 non-null   float64
 10  vote_count          5392 non-null   float64
 11  revenue             5392 non-null   int64
dtypes: float64(3), int64(2), object(7)
memory usage: 547.6+ KB
```

JSON Objects

The data in some of our columns was in the form of a JSON object, which was a problem for us. In columns such as genre, each row has a value for the genre's id and name.

```
Out[4]: 0        [{'id': 16, 'name': 'Animation'}, {'id': 35, '...
        1        [{'id': 12, 'name': 'Adventure'}, {'id': 14, '...
        2        [{'id': 10749, 'name': 'Romance'}, {'id': 35, ...
        3        [{'id': 35, 'name': 'Comedy'}, {'id': 18, 'nam...
        4                          [{'id': 35, 'name': 'Comedy'}]
                                    ...
        45461    [{'id': 18, 'name': 'Drama'}, {'id': 10751, 'n...
        45462                       [{'id': 18, 'name': 'Drama'}]
        45463    [{'id': 28, 'name': 'Action'}, {'id': 18, 'nam...
        45464                                                   []
        45465                                                   []
        Name: genres, Length: 45466, dtype: object
```

This makes it difficult to access the values, so we decided to convert columns such as genres into two columns of its elements: genres_id and genres_name. To do this we then decided to convert the JSON object into lists.

```
# genres
data_genres = combined_df["genres"]
data_genres_df = pd.DataFrame(data_genres)
data_genres_df

genres_ids = data_genres_df['genres'].apply(lambda x: [entry['id'] for entry in ast.literal_eval(x)])
genres_names = data_genres_df['genres'].apply(lambda x: [entry['name'] for entry in ast.literal_eval(x)])
```

- Make a data frame of the genres column.
- Create two separate lists from said data frame.
- We proceeded to add new columns to our actual data frame.
- Did this for all columns with values inside JSON objects. (We also converted release_date [01-02-1995] into release_year [1995], we only need the year).

```
# new columns in combined_df

# from:
# production_companies
combined_df["production_companies_id"] = prod_comps_ids
combined_df["production_companies_name"] = prod_comps_names

# genres
combined_df["genres_id"] = genres_ids
combined_df["genres_name"] = genres_names

# production_countries
combined_df["production_countries_id"] = prod_countries_id
combined_df["production_countries_name"] = prod_countries_names

# release_date
combined_df["release_year"] = release_years
```

Incorrect Data Types

There are some columns in the data frame that are in the incorrect data type. These include the “budget” and “revenue” columns. We would like to deal with these columns as integers, but the “budget” column is seen as an object and the “revenue” column is seen as a float. In order to fix this we changed the “revenue” and “budget” data type to “int64” , an integer.

```
combined_df["budget"] = combined_df["budget"].astype("int64")
combined_df["revenue"] = combined_df["revenue"].astype("int64")
```

Multiple-value rows

There are some columns which contain multiple values in their rows. This isn’t ideal as we want single-value rows to create our algorithm. These included the columns we converted from JSON objects into lists of their respective values. Such as the genres (now genres_id and genres_name) columns. ***After seeing the genres_id column had bad correlation for our model, we decided not to include it in our final dataset***.

As you can see “Drama” can appear by itself, at the start of a list, in the second position of the list, at the end of the list etc. The number of times these different sets of genres appear is usually once because the order of the different genres that can be put down is all different.

```
data.genres.value_counts()
```

```
Drama                                           3829
Comedy                                          2573
Drama, Romance                                  1090
Documentary                                     1069
Comedy, Drama                                    895
                                                ...
Comedy, Western, Action                            1
Drama, Mystery, TV Movie                           1
Action, Comedy, Drama, Music, Romance, Western     1
Drama, Thriller, Adventure, Comedy                 1
Action, Science Fiction, Thriller, Adventure       1
```

The rows that appear only once are not very useful to us, therefore we came up with two solutions, remove these rows entirely or take out and replace the list with the first item to appear, the first item being the most significant to the movie, for e.g Comedy, Western, Action -> Comedy. In the end we went with the latter solution.

Another example where this problem was apparent was the columns “production_countries” and, “production_companies” as some movies would have been produced or filmed in multiple different countries, this would have of course caused the same issue as before where the different sets of countries would contain elements that have been already recorded but would not be counted as that element and would have been recorded as an element all by itself, which wouldn’t be very useful to the machine learning model. In order to fix this problem we followed the same solution to the genre column, we replaced the list with the first country in that list as this was where most of the movie was filmed in.

```
United States of America                                        17851
United Kingdom                                                   2238
France                                                           1654
Japan                                                            1356
Italy                                                            1030
                                                                 ...
Sweden, Denmark, Germany                                            1
Argentina, Uruguay                                                  1
Paraguay                                                            1
Colombia, Mexico                                                    1
Czech Republic, Poland, Slovenia, United States of America          1
Name: production_countries, Length: 2389, dtype: int64
```

NULL Values

In our dataset of over 45,000 rows, we came across a lot of null values. Many of the null values we encountered were on our “budget” and “revenue” columns. This was a problem as “budget” was our main predictor and “revenue” was our target value, both of these features were essential in our machine learning algorithm.

We were both initially concerned with dropping too many lines as we thought it may leave us with too small a dataset to work with, however we realised that it was more important to have a good set of data which did not have missing values as the missing values would more likely skew our results.

Expectedly when we dropped the rows, most of the dropped rows were foreign movies of a lower budget which the dataset hadn’t much information on, and so we were left with five thousand plus movies that were of a higher production scale.

In order to get rid of the nulls for “budget” and “revenue” we used the following code.

```
combined_df = combined_df[(combined_df["budget"] != "0") & (combined_df["revenue"] > 0)]
```

This left us with 5392 rows of data, nowhere near the size of our original data set, however the most important thing was that they were complete and did not have any missing values.

```
combined_df.shape
```

```
(5392, 12)
```

S2 – Algorithm Description

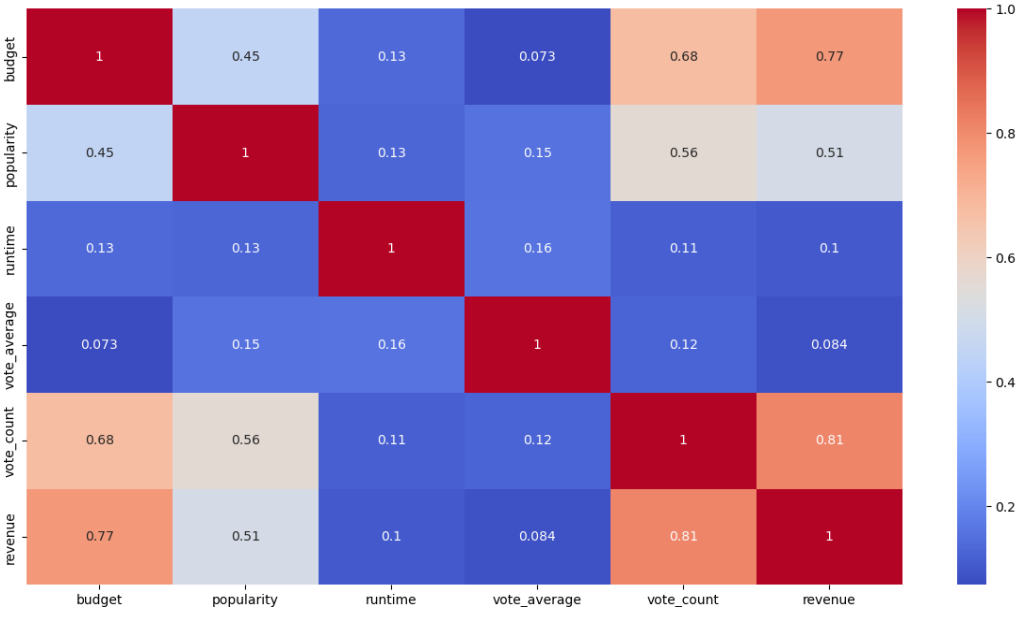
Approach: Linear Regression

After all our research we felt that the Linear Regression model was best suited to our dataset, although it has its pros and cons, we felt the pros outweigh the cons. Linear Regression models are known to be highly susceptible to outliers however we felt we had done well to clean our dataset.

We felt the Linear Regression model would work extremely well with our numerical data, especially some of our more highly correlated features such as “budget” and “vote_count”.

Before we decided to use the Linear Regression model we investigated if there was a high correlation between some of the features. Using some of the features in the seaborn library we were able to visualise the correlation coefficients between the features.

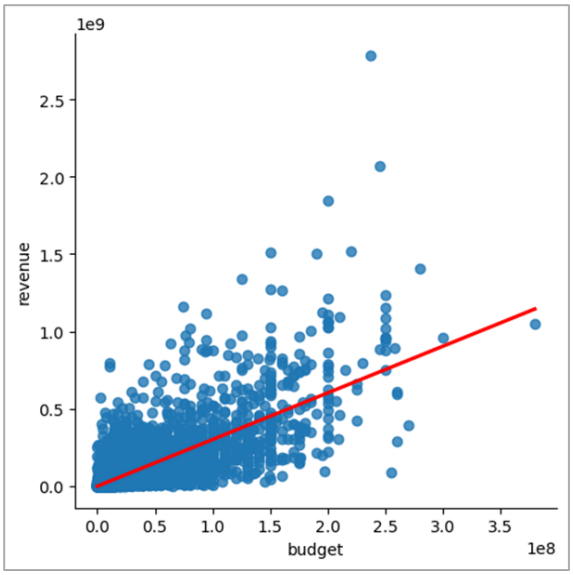
This heatmap shows the correlation coefficients between all the numerical data and our target value “revenue”. This visualisation shows the high correlation between “budget” and “revenue”, meaning the more money spent on a film the more likely it was to be successful (success based on revenue)

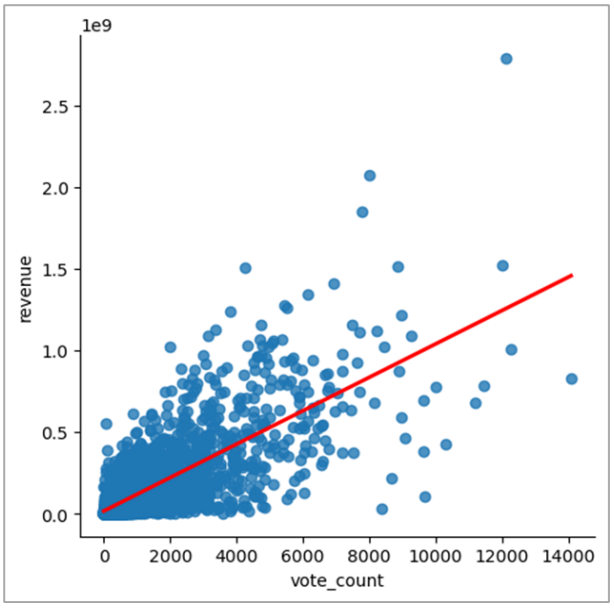


Linear Regression Graph

This visualisation shows the linear regression line between the feature “budget” and “revenue”. The closer the points to the regression line, the better the correlation.

The linear regression x-axis (budget) goes up to regression line then sees the revenue at the y-axis





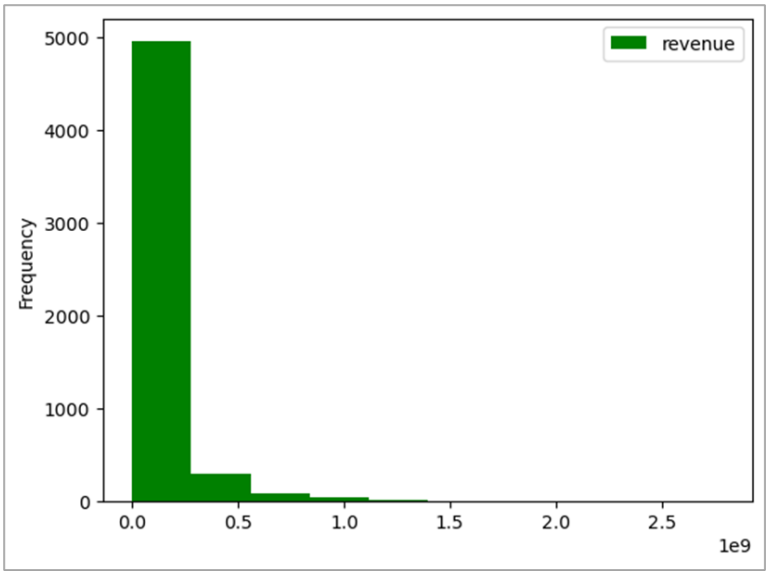
Linear Regression Graph

Linear regression: (vote_count, revenue), this visualisation shows the linear regression line between the feature “vote_count” and “revenue”.

Again, the closer that the points are to the regression line the better the correlation.

Histogram of Revenue Frequency

This histogram shows where most of the movie's “revenue” lies. Most movies made between 0 and 500 million dollars.



Split Data 80/20

First off, to split and train our model we will be using the scikit-learn machine learning library (sklearn)

We will be splitting the data 80/20 – in that 80% of the data will be used to train the model and 20% will be used to test the model. This is done because if we were to test the model on the same data we used to train it, we would be giving it the answers beforehand.

```
from sklearn.model_selection import train_test_split

train, test = train_test_split(combined_df, test_size=0.2)
```

```
train.shape
```

```
(4313, 12)
```

```
test.shape
```

```
(1079, 12)
```

- Train data: 4313 rows, 12 columns
- Test data: 1079 rows, 12 columns

Training the Model

```
from sklearn.linear_model import LinearRegression

reg = LinearRegression()

predictors = ["budget", "vote_count"]
target = "revenue"

reg.fit(train[predictors], train["revenue"])

predictions = reg.predict(test[predictors])
```

We trained the Linear Regression model using functions from scikit-learn

S3 – Results:

After training our model we got our results, and they looked promising. When we first got the results, they were in the form of scientific numbers. i.e., to the power of 10. This was not ideal as when added to the data frame they came out like this, and it was not what we wanted. They were also in floating points, and we were only looking for the whole numbers as these numbers were/are very large, some in the 100s of millions. We were able to do this in the same code, turning the predictions which were in a NumPy array into a list.

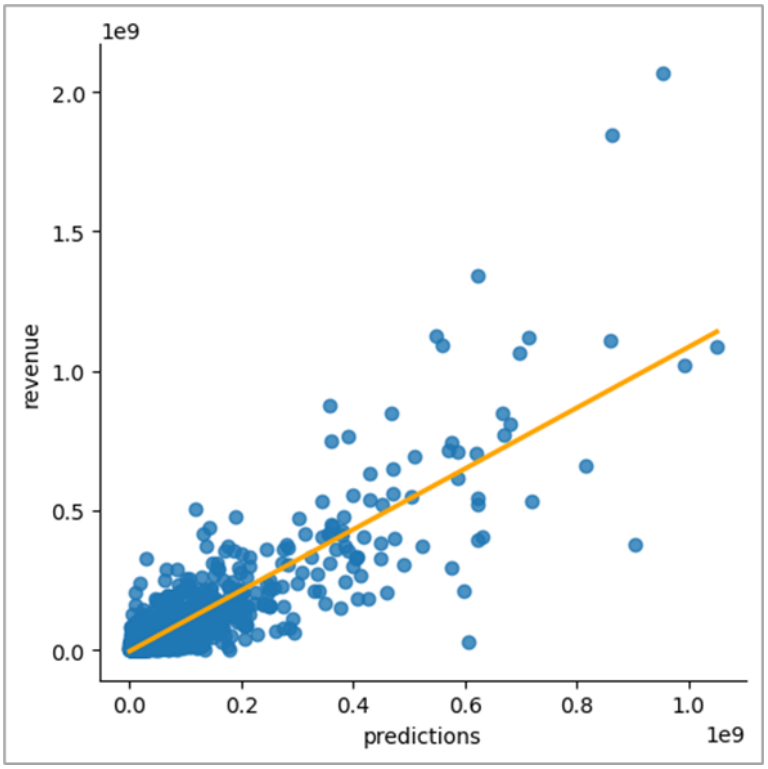
```
import numpy as np

def format_predictions(arr):
    return [f'{x:.0f}' for x in arr]

predictions = format_predictions(predictions)

predictions = [int(each) for each in predictions]
```

S3 – Actual vs Predictions



Most values are close to the regression line. Pretty accurate. Some outliers. In the high end. Certain massive movies which flopped and certain movies which were massive hits. Perhaps Avatar, Avengers etc. Still massive predictions of $1billion but they did exceptionally well and made $2billion.

Each blue point represents a movie which at the x-axis is the predicted revenue of said movie and at the y-axis, it is its actual revenue. The closer the blue points (movies) are to the regression line the more accurate our model has shown to be. As a large amount of the blue points are grouped around the regression line, we can say our model gave us near accurate predictions.

S3 – Analysis: Error Metric

RMSE: Root Mean Squared Error

```
from sklearn.metrics import mean_squared_error

error1 = mean_squared_error(test["revenue"], test["predictions"], squared=True)
error1
```

```
7535844274316178.0
```

The RMSE method does not suit our model. RMSE is a great error metric to use but it does not take kindly to outliers. Our values we see in our model have a huge range which don't help with this method. Some movies make $500,000, while others make $1,000,000,000. This massive range means that essentially all our figures are outliers as there's just such a great gap between them.

$$\mathrm{RMSD} = \sqrt{\frac{\sum_{i=1}^{N} (x_i - \hat{x}_i)^2}{N}}$$

RMSD = root-mean-square deviation
i = variable i
N = number of non-missing data points
x_i = actual observations time series
$\hat{x}_i$ = estimated time series

MAPE: Mean Absolute

```
from sklearn.metrics import mean_absolute_percentage_error

M_PER = mean_absolute_percentage_error(test["revenue"], test["predictions"])

M_PER
```

```
25054.537857443676
```

Percentage Error:

Again with MAPE (mean absolute percentage error) it does not take kindly to outliers, and like explained earlier our model has a very wide range of values which do not help. The figure here represents that on average our predictions were off by a factor of 250 times, which is not correct, as seen in our linear regression plot between our predicted revenue and actual revenue.

$$M = \frac{1}{n} \sum_{t=1}^{n} \left| \frac{A_t - F_t}{A_t} \right|$$

M = mean absolute percentage error
n = number of times the summation iteration happens
A_t = actual value
F_t = forecast value

MAE: Mean Absolute Error

```
from sklearn.metrics import mean_absolute_error

M_ABS = mean_absolute_error(test["revenue"], test["predictions"])

M_ABS
```

```
45655796.56441149
```

We believe that MAE (mean absolute error) is the best error metric for us to use. MAE gives us a figure which represents on average how far off our predictions were from the actual figure. Average difference between our predicted and actual values is $45.6million. The result provides us with a more accurate representation of our results given our huge range. MAE is less sensitive to outliers, which was a major problem in the previous metrics. While not perfect, and this may not seem good for a film which makes $1million, for big films that earn upwards of hundreds of millions it seems fair. If a film makes $300m and we predicted $345.6m, it's pretty accurate.

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$$

MAE = mean absolute error

y_i = prediction

x_i = true value

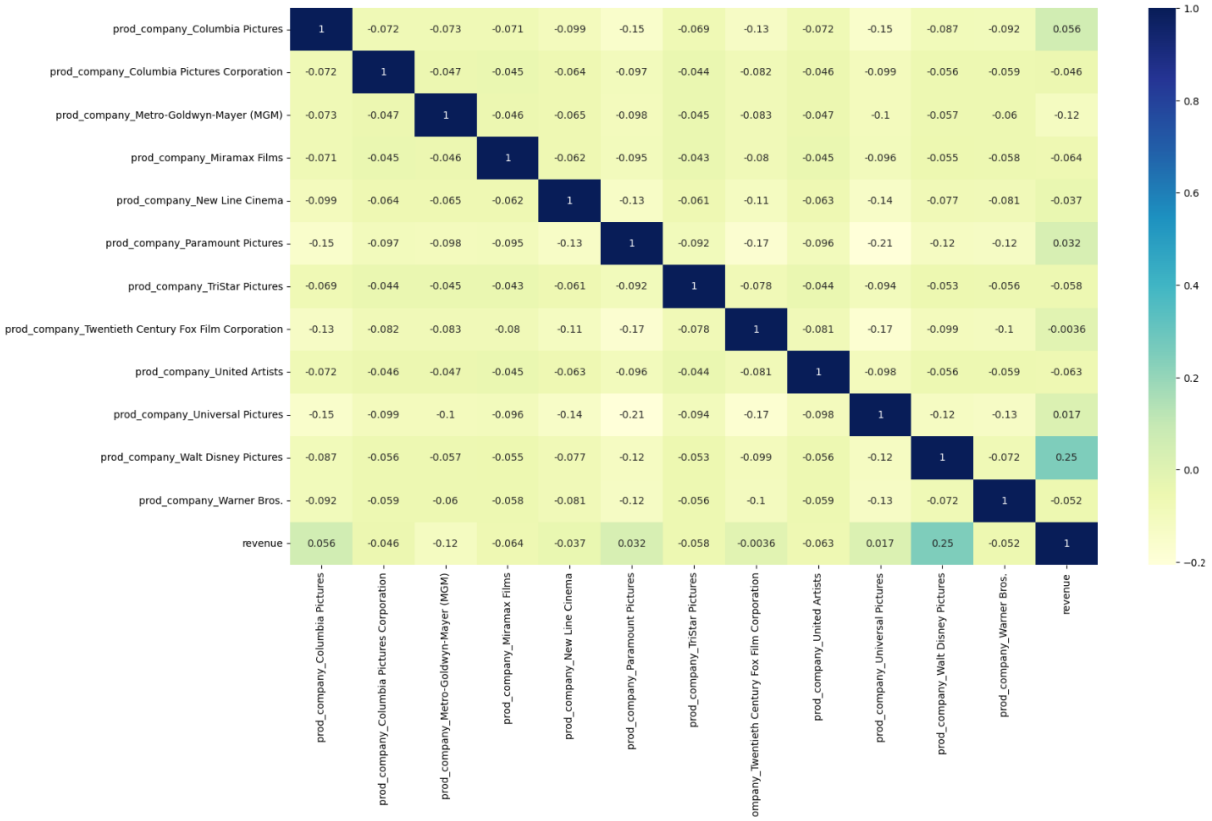
n = total number of data points

S3 – Further Analysis and Insights

We had also done some further analysis when it came to some of the individual features such as “genres”, “”production_country”, and “production_company”. We investigated the effects of each individual category and showed our findings through a very simple heatmap which shows the correlation coefficients.

Production Companies

In this first analysis of the “production_companies” column we investigated each individual production company. Because there were so many different production companies, we only took the 12 that appeared most frequently in order for the correlation to be less skewed.



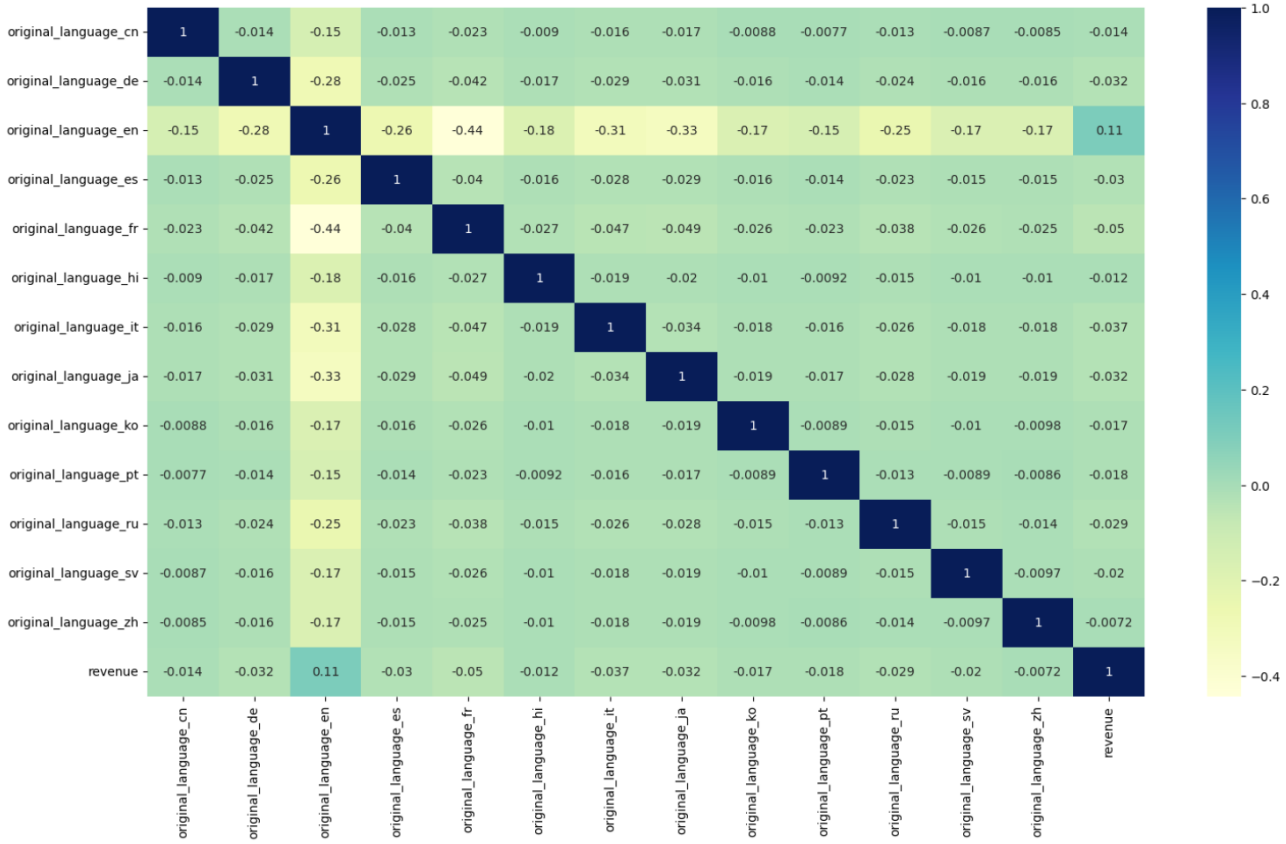
In conclusion we found that movies that were specifically produced by “Walt Disney Pictures” were a lot more likely to generate a higher revenue and therefore be more successful than all the other production companies.

This conclusion made sense however we believe that some of the correlation coefficients were skewed most likely due to the fact that our dataset was not perfect and may not have included many successful movies produced by certain production companies therefore skewing the results.

However, it’s worth noting that since we only took the first production company, if say a movie had been produced by Columbia and Disney, Columbia would only be recognised as its first in the list.

Movie Original Language

We also investigated each original language and only included the languages with the most appearances in order to make the analysis more fair.



As you would expect, the language with the highest correlation coefficient with respect to revenue would be “english” as shown above on the heatmap.

The correlation coefficients above are most likely not completely accurate and so we believe that this is due to an unfair bias in our data where most of the movies in our dataset were in english, this would most definitely cause a skew in the data.

Genres

Another feature we investigated was the genres of the movies, as you can see the genre with the strongest correlation with revenue is “Adventure”. In conclusion movies with the genre “Adventure” are more likely to be generate more revenue

Conclusion

We thought our predictions would be a bit more accurate and that our features would have a stronger correlation. Our model was most likely skewed by categorical data like genres, production companies and production countries.

Our predictions could have gone wrong for various reasons however most of them are likely to be related to issues within the dataset. Having a big range in our values had a significant impact on our model’s predictions and error metrics. Underfitting may have occurred, where the model was essentially too simple to be able to recognise patterns in the data. In order for us to improve our model it may require us to find a new dataset, a dataset which would have to be much larger and contain less null values. Maybe using a different algorithm could have been more suitable for this dataset.

Overall we were quite happy with the way things turned out, the values that we predicted using the model were quite close to the actual values, maybe with a better dataset and a different approach to the categorical features we could have gotten a lot closer.