INFO7374 Algorithmic Digital Marketing

Summary	This app will help movie investors make wiser decisions before they spend money, by predicting the success of the movie.
Author	Yifu Liu and Pengfei He
App URL	https://quiet-reef-82062.herokuapp.com/

Objectives

Target User:

Movie investors.

Goal

We want to help movie investors make wiser decisions before they spend money. When the investors put some properties of the movie they proposed, like genres, budgets, product companies.

The app will:
1 Return similar movies released before as references
2 Predict whether the movie will be successful or not.

The definition of success is **box is 1.5 times larger than budget**.

TMDB Dataset and Preprocessing

Dataset:

We used TMDB API and the scraping (open source python script) to get the TMDB dataset (we decided not to use IMDB because IMDB dataset recently has some restriction on the rights to use). Rows number: `1934374`

https://drive.google.com/file/d/1mnH9UaaXZ-gP3At0Q2Qus-Gz0EJOqiYx/view?usp=sharing
https://drive.google.com/file/d/19C8m6CwRu9I-eydnTbp4gPCp_eeFMb7t/view?usp=sharing

```python
def make_request(call_url, prior_attempts=0):
    if prior_attempts >= MAX_ATTEMPTS:
        return None
    response = requests.get(call_url)
    if was_rate_limited(response):
        sleep(RATE_LIMITER_DELAY_SECONDS)
    sleep(1 / MAX_DOWNLOADS_PER_SECOND)
    if was_successful(response):
        return response.json()
    else:
        sleep(1)  # attempt to sleep through any intermittent issues
        return make_request(call_url, prior_attempts + 1)


def make_detail_request(category, entry_id):
    category_specifics = ''
    if category in CATEGORY_SPECIFIC_CALLS:
        category_specifics = CATEGORY_SPECIFIC_CALLS[category]
    call_url = BASE_API_CALL.format(
        category=category,
        entry_id=entry_id,
        api_key=API_KEY,
        category_specifics=category_specifics,
                                    )
    return make_request(call_url)
```

Preprocessing:

Data preprocessing includes drop duplicated values, drop null values, and drop meanlingless columns:

```python
[ ] # Drop Duplication Values
    duplicated_values = ('num_voted_users', 'popularity', 'budget', 'genres', 'id', 'plot_keywords', 'language', 'original_title','overview', '
    original_format.drop_duplicates(subset=duplicated_values, keep='first', inplace=True)
```

```python
[ ] # Homepage and tagline are useless in our model
    original_format.drop(['homepage','tagline'], axis=1, inplace=True)
```

```python
[ ] # Drop rows with invalid voting score
    original_format = original_format[original_format['vote_average'] > 0]
```

```python
[ ] # Drop rows with invalid duration
    original_format = original_format[10 < original_format['duration']]
    original_format = original_format[original_format['duration'] < 300]
```

```python
[ ] # Drop rows with invalid budget
    original_format.budget=original_format.budget.astype(int)
    original_format = original_format[original_format.budget!=0]
```

Categorize columns:

Based on the discussion with the Professor in class, we decide to numeralize and give a threshold for each column, so that we can make the similar search in a more clear way. For example, we only allow top ten producing companies in the dataset, all other companies will fall in 'other company'. Thus, when we ask users to put into the company name, they are only allowed to select from these 10 + 1 names. And the similarity search will then become very clear, since there are very clear 11 categories to choose from.

Dataset before:

```
original_format.tail(8)
```

	budget	genres	homepage	id	plot_keywords	language	original_title	overview	popularity
1934367	0	Action|Animation	NaN	74616	martial arts|street fighter|based on video game	日本語	スーパーストリートファイターIV	Chun-Li, Guile and Cammy must track down Juri,...	1.121
1934368	0	Animation|Action|Adventure	NaN	74617		日本語	マジンガーZ対暗黒大将軍	Kouji and his friends have defeated Dr. Hell a...	1.638
1934369	0	Horror	NaN	74618		English	Blitzkrieg: Escape from Stalag 69	Enter Stalag 69, where torture is just the Beg...	1.723
1934370	0	Animation|Adventure|Action	NaN	74619	anime	日本語	マジンガーＺ対デビルマン	Mazinger Z vs. Devilman is a 1973 animated mov...	1.575
1934371	0	Action|Thriller	NaN	74620		English	Hollywood Flies	While on a road trip, a man and his sister pic...	1.684
1934372	0	Animation|Action|Adventure|Science Fiction	NaN	74621	super robot|go nagai	日本語	グレートマジンガー対ゲッターロボ	When an UFO delivers a metal-eating monster to...	1.018
1934373	0	Animation|Action|Science Fiction	NaN	74622	ufo|anime|super robot|go nagai	日本語	グレートマジンガー対ゲッターロボG 空中大激突	When a UFO arrives from space and attacks the ...	1.84
1934374	0	Action|Adventure|Animation	NaN	74623	giant robot|giant monster|kaiju	日本語	グレンダイザー・ゲッターロボG・グレートマジンガー 決戦！大海獣	A giant sea monster known as Dragonsaurus has ...	1.072

	production_companies	production_countries	release_date	box_office	duration	spoken_languages	status	tagline	movie_title	vote_average	n
1934367	[{'id': 1077, 'logo_path': '/1tpNgC6BptieT9oF2...	[{'iso_3166_1': 'JP', 'name': 'Japan'}]	2010-04-27	0.0	35.0	[{'iso_639_1': 'ja', 'name': '日本語'}]	Released	NaN	Super Street Fighter IV	6.8	
1934368	[{'id': 5542, 'logo_path': '/ayE4LIqoAWotavo7x...	[{'iso_3166_1': 'JP', 'name': 'Japan'}]	1974-07-25	0.0	43.0	[{'iso_639_1': 'ja', 'name': '日本語'}]	Released	Deep beneath the earth, seven robots lay dorma...	Mazinger Z vs The Great Dark General	5.6	
1934369	[]	[{'iso_3166_1': 'US', 'name': 'United States o...	2008-03-01	0.0	120.0	[{'iso_639_1': 'en', 'name': 'English'}]	Released	Stalag 69: Where Nazis Rule With Perversion An...	Blitzkrieg: Escape from Stalag 69	5.0	
1934370	[{'id': 5542, 'logo_path': '/ayE4LIqoAWotavo7x...	[{'iso_3166_1': 'JP', 'name': 'Japan'}]	1973-07-18	0.0	43.0	[{'iso_639_1': 'ja', 'name': '日本語'}]	Released	NaN	Mazinger Z vs. Devilman	4.8	
1934371	[{'id': 1069, 'logo_path': None, 'name': 'GFT ...	[{'iso_3166_1': 'CA', 'name': 'Canada'}, {'iso...	2005-06-28	0.0	95.0	[{'iso_639_1': 'en', 'name': 'English'}, {'iso...	Released	NaN	Hollywood Flies	5.3	
1934372	[{'id': 5542, 'logo_path': '/ayE4LIqoAWotavo7x...	[{'iso_3166_1': 'JP', 'name': 'Japan'}]	1975-04-21	0.0	30.0	[{'iso_639_1': 'ja', 'name': '日本語'}]	Released	NaN	Great Mazinger vs. Getter Robo	5.1	
1934373	[{'id': 5542, 'logo_path': '/ayE4LIqoAWotavo7x...	[{'iso_3166_1': 'JP', 'name': 'Japan'}]	1975-07-26	0.0	25.0	[{'iso_639_1': 'ja', 'name': '日本語'}]	Released	NaN	Great Mazinger vs. Getter Robo G: The Great Sp...	5.6	
1934374	[{'id': 12268, 'logo_path': None, 'name': 'Dyn...	[{'iso_3166_1': 'JP', 'name': 'Japan'}]	1976-07-18	0.0	31.0	[{'iso_639_1': 'ja', 'name': '日本語'}]	Released	NaN	Grendizer, Getter Robo G, Great Mazinger: Deci...	6.3	

Dataset after:

```
[44] original_format.head()
```

War	Music	Western	Mystery	History	Horror	TV Movie	Steven Spielberg	Clint Eastwood	Woody Allen	Martin Scorsese	Ridley Scott	Brian De Palma	Steven Soderbergh	Wes Craven
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Francis Ford Coppola	Ron Howard	Other Director	Universal Pictures	Paramount	Columbia Pictures	Warner Bros. Pictures	20th Century Fox	New Line Cinema	Walt Disney Pictures	Touchstone Pictures	Miramax	Metro-Goldwyn-Mayer	Other Company
0	0	1	0	0	0	0	0	0	0	0	1	0	0
0	0	1	1	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	1

Budget as example:

```
[ ] # For budget column
    original_format.budget=original_format.budget.astype(int)
```

```
[ ] # Create threshold to decide five levels of budget: VeryLowBudget, LowBudget, MedBudget, HighBudget, VeryHighBudget
    # After this step, each new movie's budget will only fall in one of them. Thus, we can have a much clear way to make
    # similar seach.
    level_1 = original_format.budget[original_format.budget>0].quantile(0.25)
    level_2 = original_format.budget[original_format.budget>0].quantile(0.5)
    level_3 = original_format.budget[original_format.budget>0].quantile(0.75)
    level_4 = original_format.budget[original_format.budget>0].quantile(0.95)
```

```
[ ]
    original_format['VeryLowBudget'] = original_format['budget'].map(lambda s: 1 if 0< s < level_1 else 0)
    original_format['LowBudget'] = original_format['budget'].map(lambda s: 1 if level_1 <= s < level_2 else 0)
    original_format['MedBudget'] = original_format['budget'].map(lambda s: 1 if  level_2 <= s < level_3 else 0)
    original_format['HighBudget'] = original_format['budget'].map(lambda s: 1 if level_3 <= s < level_4 else 0)
    original_format['VeryHighBudget'] = original_format['budget'].map(lambda s: 1 if s >= level_4 else 0)
```

```
[ ] # Similarly, we also seperate the length of movie into three levels
    original_format['ShortMovie'] = original_format['duration'].map(lambda s: 1 if s < 90 else 0)
    original_format['NormalMovie'] = original_format['duration'].map(lambda s: 1 if 90 <= s < 120 else 0)
    original_format['LongMovie'] = original_format['duration'].map(lambda s: 1 if s >= 120 else 0)
```

Algorithm learned from class: Decision Tree

The algorithm we use to make predictions is the decision tree. We used the default model from sklearn. The accuracy score we get is **0.7**.

6 Algorithm learned from Class: Decision Tree

```
# Split data into training and testing data set
y = original_format['success']
y = np.array(y).reshape(-1,1)
x = original_format.drop('success',axis=1)
```

```
[50] x_train_all, x_test, y_train_all, y_test = train_test_split(x,y,random_state=10, test_size=.15)
     x_train, x_valid, y_train, y_valid = train_test_split(x_train_all, y_train_all, random_state=10, test_size=0.15)
```

```
[51] decision_tree = DecisionTreeClassifier(criterion='entropy',max_depth=20, min_samples_leaf=10)
     decision_tree.fit(x_train, y_train)
```

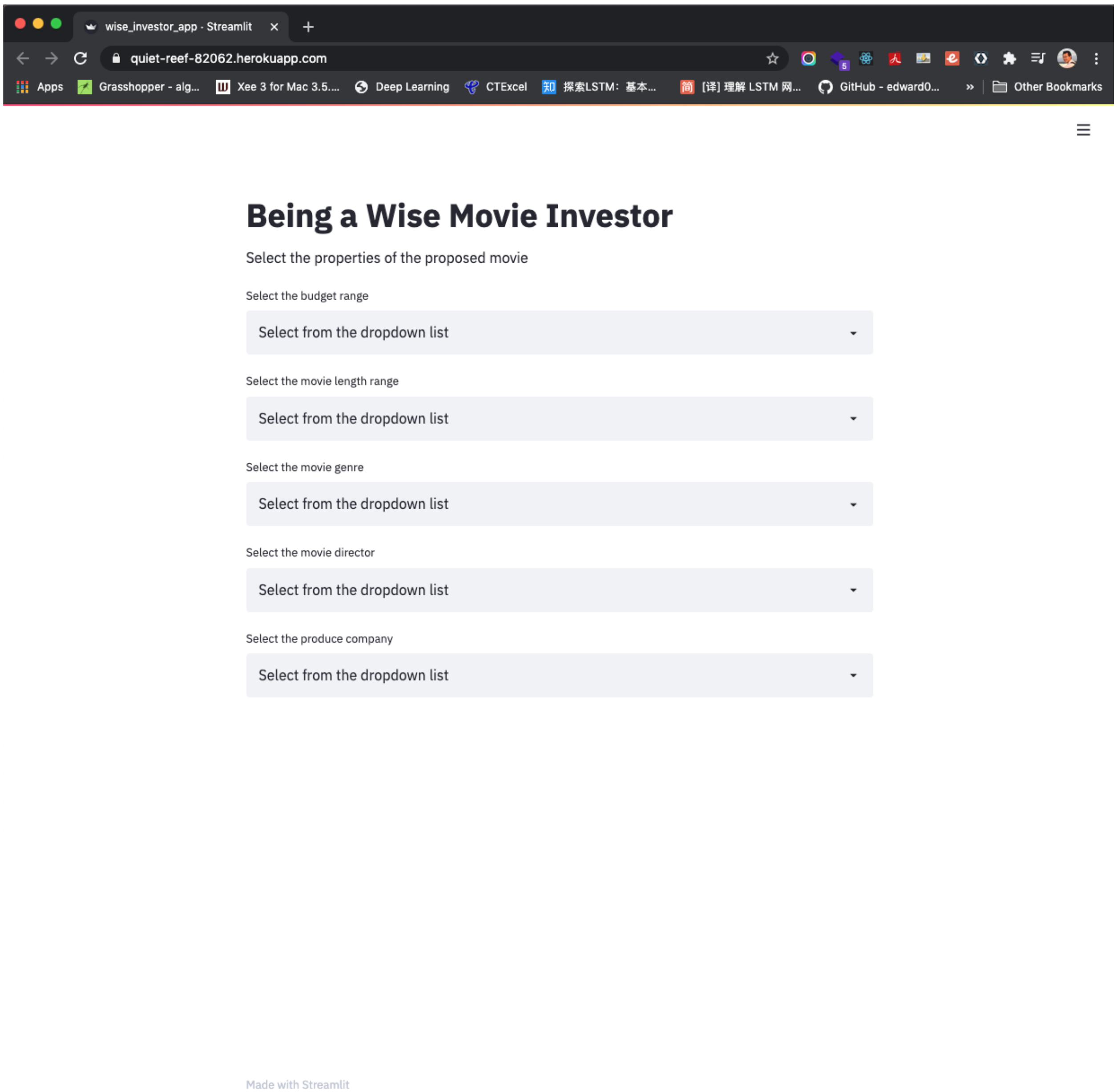
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=20, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=10, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

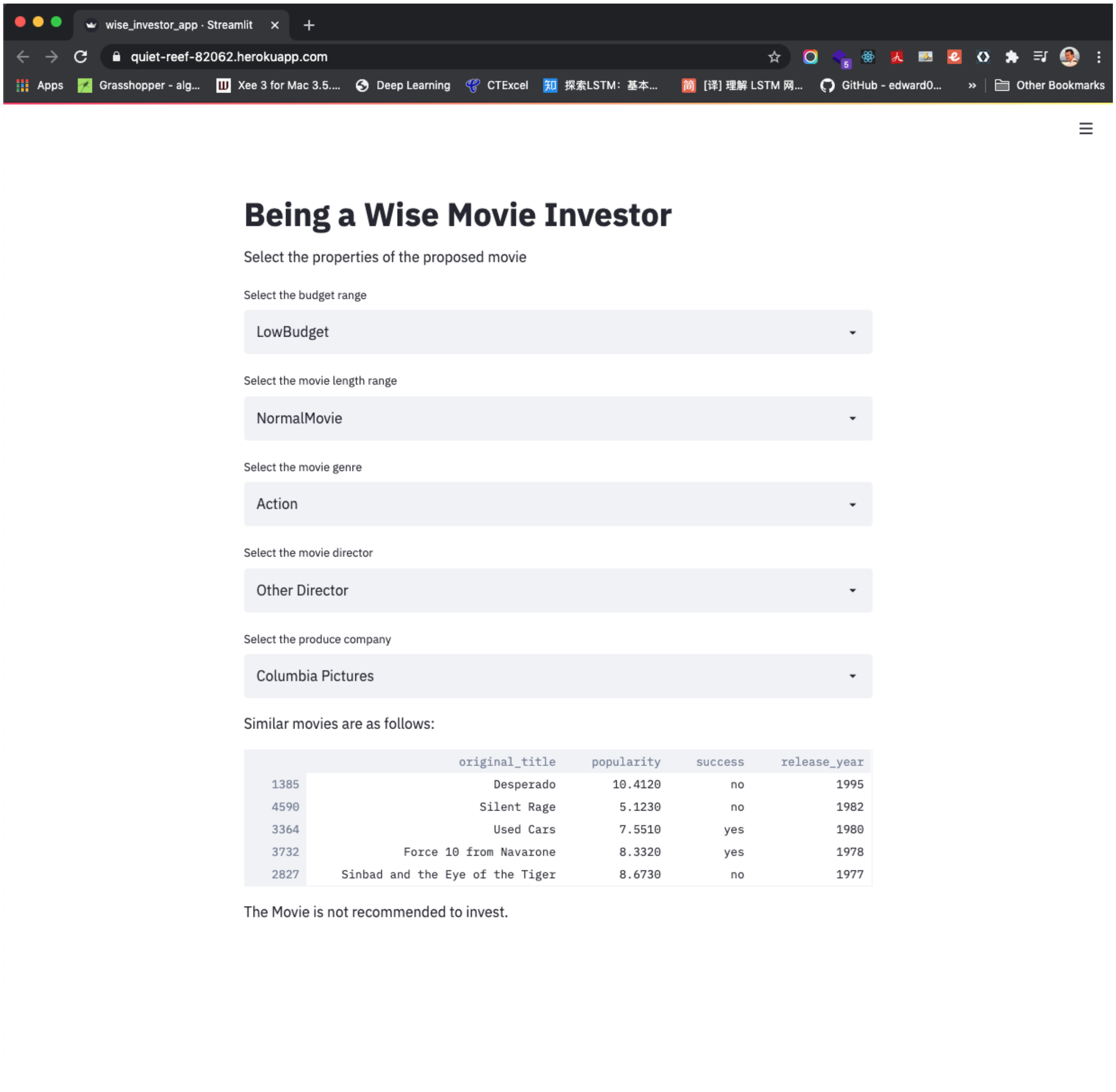
```
[52] score_all=cross_val_score(decision_tree, x_train_all, y_train_all, cv=5)
     avg_score=score_all.mean()
     print("Accuracy Score based on 5-fold Cross Validation: {}\n".format(round(avg_score,2)))
```

```
Accuracy Score based on 5-fold Cross Validation: 0.7
```

Being a Wise Movie Investor App

The application is deployed on Heroku: https://quiet-reef-82062.herokuapp.com/





Fast API

As we discussed with the professor in the last class. We can only use streamlit for our application. But we want to demonstrate the knowledge we learned from this class as much as possible. So we also create a version of application which integrate streamlit with fastapi.

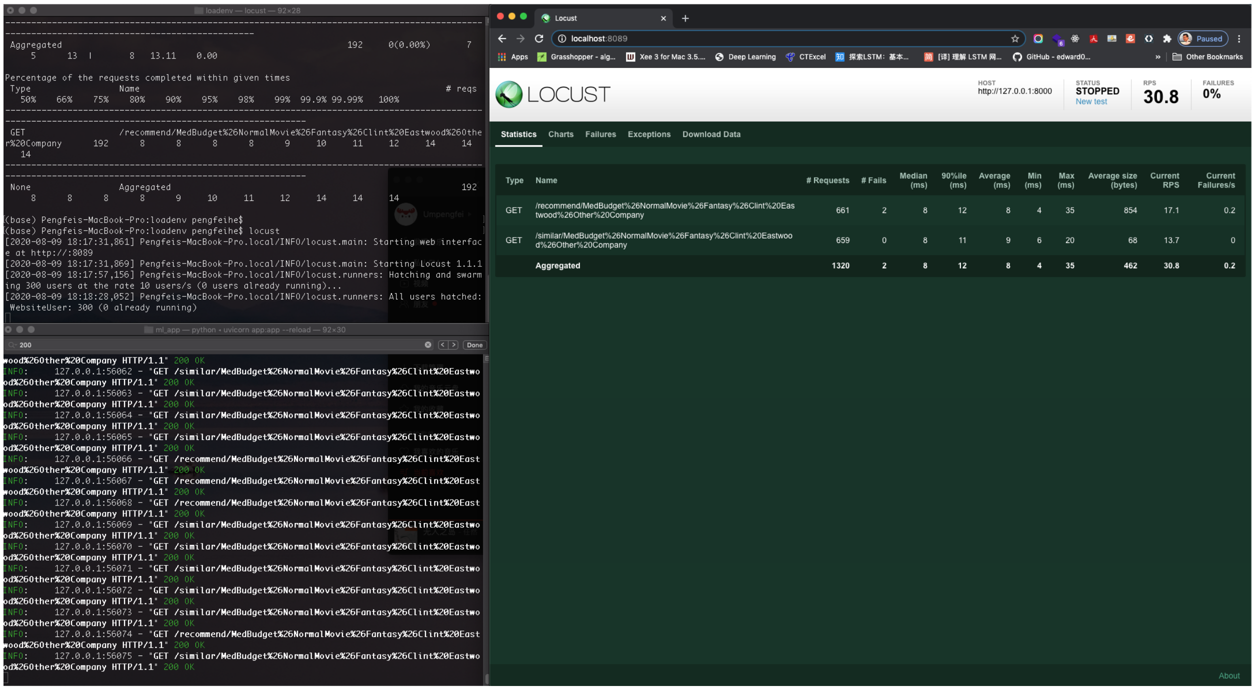
```python
@app.get('/similar/{query}')
async def get_npf(query):
    params = query.split('&')
    budget = params[0]
    length = params[1]
    genre = params[2]
    director = params[3]
    company = params[4]
    df = similar_data.loc[(similar_data[budget] == 1) & (similar_data[length] == 1) &
                          (similar_data[genre] == 1) & (similar_data[director] == 1) & (similar_data[company] == 1)]
    df_sorted = df.sort_values('release_year', ascending=False).head(5)
    df_display = df_sorted[['original_title',
                            'popularity', 'success', 'release_year']]
    df_display['success'] = df_display['success'].map(
        lambda s: 'yes' if s == 0 else 'no')

    return df_display


@app.get('/recommend/{query}')
async def get_npa(query):
    params = query.split('&')
    budget = params[0]
    length = params[1]
    genre = params[2]
    director = params[3]
    company = params[4]
    df_pre = predict_data.loc[(similar_data[budget] == 1) & (similar_data[length] == 1) &
                              (similar_data[genre] == 1) & (similar_data[director] == 1) & (similar_data[company] == 1)]
    return df_pre
```

Load Testing using Locust

We also used locust for a load testing. We tested the FastAPI on the local machine. It is tested on 300 users, 10 user / second, each user will wait 5-15s. The result is as follows:



This api performs quite well. It finished the whole test without making a failure and the response time is very short.

Special Thank to

Professor: Srikanth Krishnamurthy

TA: Abhishek Maheshwarappa

And everyone in this wonderful summer class!