



INTELIGENCIA ARTIFICIAL

ANDREA BERRONES GARZA
M.I.C NORMANDO ALI ZUBIA HERNÁNDEZ

CHIHUAHUA, CHIHUAHUA VIERNES 31 DE MARZO DEL 2019

ÍNDICE

• Planteamiento del problema	3
• Objetivo	3
• Hipótesis	4
• Análisis del problema	4
• Pre-procesamiento.....	4
• Implementación de modelo de machine learning	13
• Conclusión	16

PLANTEAMIENTO DEL PROBLEMA

Una empresa de bienes raíces busca un método automatizado para predecir el precio de una casa en base a la información de la misma. Cuentan con una base datos con 79 características y con 1460 casas registradas.

OBEJETIVO:

Encontrar el precio de venta de las casas.

CARACTERÍSTICAS

Característica	Tipo	Descripción
Id	Numerico	Identificador
belongs_to_collection	Nominal	Pertenece a alguna saga
Budget	Numerico	Presupuesto
Genres	Nominal	Generos de la película
Homepage	Nominal	Pagina de internet
imdb_id	Nominal	Identificador en imbd
original_language	Nominal	Idioma original de la película
original_title	Nominal	Título original de la película
Overview	Nominal	Sinopsis
Popularity	Numerico	Popularidad
poster_patch	Nominal	Poster
production_companies	Nominal	Compañías que producen la película
production_countries	Nominal	País de producción
realise_date	Date	Fecha de lanzamiento
runtime	Numerico	Duración
spoken_languages	Nominal	Idiomas
status	Nominal	Estado de lanzamiento/producción
tagline	Nominal	Frase clave
title	Nominal	Título
Keywords	Nominal	Palabras clave
Cast	Nominal	Elenco
crew	Nominal	Equipo
revenue	Numerico	Ingresos(esta es la clase)

HIPÓTESIS

Las características que considero más importantes para las ganancias de las películas son las siguientes:

- Genres, el genero de la película es importante por que el tamaño de la audiencia cambia dependiendo de que tipo es.
- Runtime, la duración de la película, cuando son muy largas a muchas personas no les gusta

- Spoken_languages, si no esta en ingles es probable que sea menos conocida.
- Overview, La sinopsis es importante para atraer al público.
- Cast, hay actores que tienen mucha fama y mucho público.
- Crew, hay directores que atraen más audiencia que otros.
- Popularity, entre más popular más gente la ve y más ganancia.

Lo primero que hice fue importar todas las librerías que iba a usar en este problema e importar los datos del csv.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import ast
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn import linear_model
from sklearn.ensemble import BaggingRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.preprocessing import MultiLabelBinarizer, LabelEncoder
```

```
data = pd.read_csv("train.csv")
```

```
data.shape
```

```
(3000, 23)
```

```
data.info()
```

Seguido de eso obtuve la información de los datos, pude notar que la mayoría de los atributos son de texto, no numéricos que hay unas clases con muchos registros vacíos, cuento con 3,000 registros para entrenar el modelo, 22 atributos y una clase.

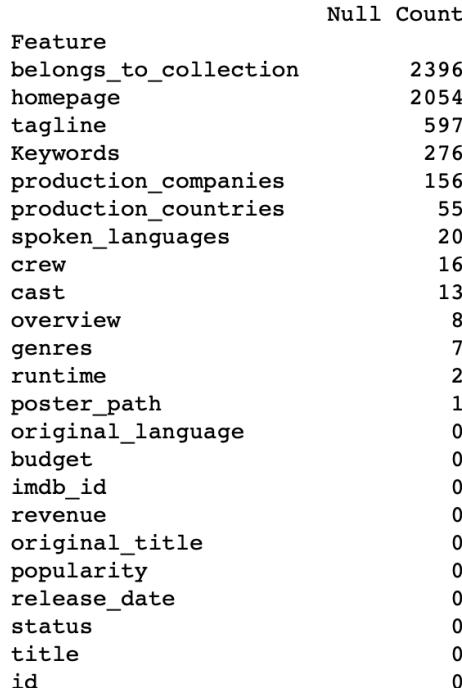
```
data.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 23 columns):
id                3000 non-null int64
belongs_to_collection 604 non-null object
budget             3000 non-null int64
genres              2993 non-null object
homepage            946 non-null object
imdb_id             3000 non-null object
original_language   3000 non-null object
original_title      3000 non-null object
overview             2992 non-null object
popularity           3000 non-null float64
poster_path          2999 non-null object
production_companies 2844 non-null object
production_countries 2945 non-null object
release_date         3000 non-null object
runtime              2998 non-null float64
spoken_languages     2980 non-null object
status               3000 non-null object
tagline               2403 non-null object
title                3000 non-null object
Keywords              2724 non-null object
cast                  2987 non-null object
crew                  2984 non-null object
revenue               3000 non-null int64
dtypes: float64(2), int64(3), object(18)
memory usage: 539.1+ KB
```

Imprimí un .head y note que varias de los atributos contienen datos de tipo JSON

id	belongs_to_collection	budget	genres	homepage	imdb_id	original_language	original_title	overview	popularity	release_date	runtime	spoken_languages	status	tagline	title	Keywords	cast	crew	revenue
0 1	[{"id": 313576, "name": "Hot Tub Time Machine"}]	14000000	[{"id": 35, "name": "Comedy"}]	NaN	tt2637294	en	Hot Tub Time Machine 2	When Lou, who has become the father of the...	6.575393	2/20/15	93.0	[{"iso_639_1": "en", "name": "English"}]	Released	The Laws of Space and Time are About to be Violated...	Hot Tub Time Machine 2	[{"id": 4379, "name": "time travel"}, {"id": 9...]	[{"cast_id": 4, "character": "Lou", "credit_id": 59ac067c9251410faf02c8d8}, {"de...	12314651	
1 2	[{"id": 107674, "name": "The Princess Diaries"}]	40000000	[{"id": 35, "name": "Comedy"}, {"id": 18, "name": "Romantic"}]	NaN	tt0368933	en	The Princess Diaries 2: Royal Engagement	Mia Thermopolis is now a college graduate and ...	8.248895	8/6/04	113.0	[{"iso_639_1": "en", "name": "English"}]	Released	It can take a lifetime to find true love; she ...	The Princess Diaries 2: Royal Engagement	[{"id": 2505, "name": "coronation"}, {"id": 42...]	[{"cast_id": 1, "character": "Mia Thermopolis"}, {"de...	52fe43fe9251416c7502563d, 95149435	
2 3	NaN	3300000	[{"id": 18, "name": "Drama"}]	http://sonyclasics.com/whiplash/	tt2582802	en	Whiplash	Under the direction of a ruthless instructor, ...	64.299990	10/10/14	105.0	[{"iso_639_1": "en", "name": "English"}]	Released	The road to greatness can take you to the edge.	Whiplash	[{"id": 1416, "name": "jazz"}, {"id": 1923, "name": "Andrew Neumann"}]	[{"cast_id": 5, "character": "Andrew Neumann"}, {"de...	54d5356ec3a3e83sa0000039, 13092000	
3 4	NaN	1200000	[{"id": 53, "name": "Thriller"}]	http://kahaaninthefilm.com/	tt1821480	hi	Kahaani	Vidya Bagchi (Vidya Balan) arrives in Kolkata ...	3.174936	3/9/12	122.0	[{"iso_639_1": "en", "name": "English"}]	Released	NaN	Kahaani	[{"id": 10092, "name": "mystery"}, {"id": 1054...]	[{"cast_id": 1, "character": "Vidya Bagchi"}, {"de...	52fe48779251416c91086eb, 16000000	
4 5	NaN	0	[{"id": 28, "name": "Action"}, {"id": 53, "name": "Thriller"}]	NaN	tt1380152	ko	마린보이	Marine Boy is the story of a former national a...	1.148070	2/5/09	118.0	[{"iso_639_1": "ko", "name": "한국어(조선文中)"}]	Released	NaN	Marine Boy	NaN	[{"cast_id": 3, "character": "Chun-soo"}, {"de...	52fe464b9251416c75073d43, 3923970	

Conté la cantidad de nulos por atributo y concluí que en la columna belongs_to_collection y homepage faltaban muchos y que la mayoría de los que si tenían dato eran únicos así que concluí que era mejor llenar los nulos con 0 y los que si tenían dato con un 1 utilizando ".loc", "fillna" y "notnull".

```
nulls = pd.DataFrame(data.isnull().sum().sort_values(ascending=False) [:])
nulls.columns = ['Null Count']
nulls.index.name = 'Feature'
print(nulls)
```



```

bid= data.belongs_to_collection.unique()
len(bid)
pbid = (len(bid) * 100) / 3000
print ("El numero de valores unicos es:")
print (len(bid))
print("El Porcentaje de unicos de es:")
print (pbid)

```

El numero de valores unicos es:
423

```

: bid= data.homepage.unique()
len(bid)
pbid = (len(bid) * 100) / 3000
print ("El numero de valores unicos es:")
print (len(bid))
print("El Porcentaje de unicos de es:")
print (pbid)

```

El numero de valores unicos es:
942
El Porcentaje de unicos de es:
31

```

bid= data.imdb_id.unique()
len(bid)
pbid = (len(bid) * 100) / 3000
print ("El numero de valores unicos es:")
print (len(bid))
print("El Porcentaje de unicos de es:")
print (pbid)

```

El numero de valores unicos es:
3000
El Porcentaje de unicos de es:
100

```

bid= data.id.unique()
len(bid)
pbid = (len(bid) * 100) / 3000
print ("El numero de valores unicos es:")
print (len(bid))
print("El Porcentaje de unicos de es:")
print (pbid)

```

El numero de valores unicos es:
3000
El Porcentaje de unicos de es:
100

Con esto también decidí eliminar las columnas de Id y imbd_id ya que son un identificador y no tienen relación alguna con la clase.

```

target = data.revenue

data = data.drop(["id"], axis = 1)
data = data.drop(["imdb_id"], axis = 1)

data.loc[data["homepage"].notnull(),"homepage"] = 1
data["homepage"] = data["homepage"].fillna(0)

data.loc[data["belongs_to_collection"].notnull(),"belongs_to_collection"] = 1
data["belongs_to_collection"] = data["belongs_to_collection"].fillna(0)

```

Empecé a trabajar con los géneros ya que yo los considere un atributo importante, lo que hice fue crear una función que fuera sacando una lista de cada genero mientras hacía un “hot encode” que va transformando cada genero en una columna nueva y pone un 1 si la película cuenta con el o un 0 si no.

```

mlb = MultiLabelBinarizer()

def convertStringToList(strVal):
    if type(strVal) is not str:
        return []
    else:
        return ast.literal_eval(strVal)

def formatDictColumnAndExtractNames(strVal):
    listOfItems = convertStringToList(strVal)
    return list(map(lambda x: x['name'], listOfItems))

def extractGenres(data):
    data['genres'] = data['genres'].apply(formatDictColumnAndExtractNames)

    return data.join(pd.DataFrame(mlb.fit_transform(data.pop('genres')),
                                  columns=list(map(lambda x: 'genre_'+x, mlb.classes_)),
                                  index=data.index))

```

Volví a correr el “.info” para comprobar que se había realizado el paso anterior con éxito

<pre>data = extractGenres(data)</pre>	<pre>Keywords</pre>	2724 non-null object
<pre>data.info()</pre>	<pre>cast</pre>	2987 non-null object
	<pre>crew</pre>	2984 non-null object
	<pre>revenue</pre>	3000 non-null int64
	<pre>genre_Action</pre>	3000 non-null int64
	<pre>genre_Adventure</pre>	3000 non-null int64
	<pre>genre_Animation</pre>	3000 non-null int64
	<pre>genre_Comedy</pre>	3000 non-null int64
	<pre>genre_Crime</pre>	3000 non-null int64
	<pre>genre_Documentary</pre>	3000 non-null int64
	<pre>genre_Drama</pre>	3000 non-null int64
	<pre>genre_Family</pre>	3000 non-null int64
	<pre>genre_Fantasy</pre>	3000 non-null int64
	<pre>genre_Foreign</pre>	3000 non-null int64
	<pre>genre_History</pre>	3000 non-null int64
	<pre>genre_Horror</pre>	3000 non-null int64
	<pre>genre_Music</pre>	3000 non-null int64
	<pre>genre_Mystery</pre>	3000 non-null int64
	<pre>genre_Romance</pre>	3000 non-null int64
	<pre>genre_Science Fiction</pre>	3000 non-null int64
	<pre>genre_TV Movie</pre>	3000 non-null int64
	<pre>genre_Thriller</pre>	3000 non-null int64
	<pre>genre_War</pre>	3000 non-null int64
	<pre>genre_Western</pre>	3000 non-null int64
	<pre>dtypes: float64(2), int64(24), object(14)</pre>	
	<pre>memory usage: 937.6+ KB</pre>	

Como runtime sólo contaba con dos valores nulos decidí llenarlos con el promedio para que este atributo esté listo para usarse. Empecé a trabajar con otros atributos, cast y crew los transforme a un diccionario mientras que production_companies, production_countries y keywords los puse en una lista para poder seguir trabajando con cada uno de ellos. También cree nuevas variables contando las letras, palabras de algunos de los atributos para empezar a incluirlos al data set y poder ver su relación con la clase.

```

: data["runtime"] = data["runtime"].fillna(data["runtime"].mean())

: data.loc[data["cast"].notnull(),"cast"] = data.loc[data["cast"].notnull(),"cast"].apply(lambda x : ast.literal_eval(x))
data.loc[data["crew"].notnull(),"crew"] = data.loc[data["crew"].notnull(),"crew"].apply(lambda x : ast.literal_eval(x))

: features_to_fix=["production_companies", "production_countries","Keywords"]

for feature in features_to_fix:
    data.loc[data[feature].notnull(),feature]=\
    data.loc[data[feature].notnull(),feature].apply(lambda x : ast.literal_eval(x))\
    .apply(lambda x : [y["name"] for y in x])

: data["cast_len"] = data.loc[data["cast"].notnull(),"cast"].apply(lambda x : len(x))
data["crew_len"] = data.loc[data["crew"].notnull(),"crew"].apply(lambda x : len(x))

data["production_companies_len"] = data.loc[data["production_companies"].notnull(),"production_companies"]\
.apply(lambda x : len(x))

data["production_countries_len"] = data.loc[data["production_countries"].notnull(),"production_countries"]\
.apply(lambda x : len(x))

data["Keywords_len"] = data.loc[data["Keywords"].notnull(),"Keywords"].apply(lambda x : len(x))

data['original_title_letter_count'] = data['original_title'].str.len()
data['original_title_word_count'] = data['original_title'].str.split().str.len()
data['title_word_count'] = data['title'].str.split().str.len()
data['overview_word_count'] = data['overview'].str.split().str.len()
data['tagline_word_count'] = data['tagline'].str.split().str.len()

```

Pase los atributos de tagline, original_title y status a 1 y 0 creando tres variables nuevas y agregándolas al data set, con realise_date utilice la librería de pandas para poder separarlo en año, mes, día, día de la semana y el cuatrimestre del año. Cheque la cantidad de nulos y rellene los datos de keywords, production_companies_len, tagline_word_count, etc. Además saque una relación entre budget, realese_year y popularity.

```

data["has_tagline"] = 1
data.loc[data["tagline"].isnull(),"has_tagline"] = 0

data["title_different"] = 1
data.loc[data["title"]==data["original_title"], "title_different"] = 0

data["isReleased"] = 1
data.loc[data["status"]!="Released", "isReleased"] = 0

release_date = pd.to_datetime(data["release_date"])
data["release_year"] = release_date.dt.year
data["release_month"] = release_date.dt.month
data["release_day"] = release_date.dt.day
data["release_wd"] = release_date.dt.dayofweek
data["release_quarter"] = release_date.dt.quarter

data.loc[data["cast"].notnull(),"cast"] = data.loc[data["cast"].notnull(),"cast"]\
.apply(lambda x : [y["name"] for y in x if y["order"] < 6])

```

Feature	Null Count	
tagline_word_count	597	genre_Foreign 0
Keywords	276	revenue 0
Keywords_len	276	runtime 0
production_companies_len	156	popularity 0
production_companies	156	original_language 0
production_countries	55	homepage 0
production_countries_len	55	budget 0
crew_len	16	genre_Fantasy 0
Executive Producer	16	genre_Romance 0
Producer	16	genre_History 0
Director	16	title_word_count 0
cast	13	release_quarter 0
cast_len	13	release_wd 0
overview_word_count	8	release_day 0
genre_Crime	0	release_month 0
genre_Adventure	0	release_year 0
genre_Documentary	0	isReleased 0
genre_Comedy	0	title_different 0
genre_Animation	0	has_tagline 0
genre_Drama	0	original_title_word_count 0
genre_Family	0	genre_Horror 0
genre_Action	0	original_title_letter_count 0
		genre_Western 0
		genre_War 0
		genre_Thriller 0
		genre_TV_Movie 0
		genre_Science_Fiction 0
		genre_Mystery 0
		genre_Music 0
		belongs_to_collection 0

```

to_empty_list=["Keywords", "production_companies", "production_countries", \
    "Director", "Producer", "Executive Producer", "cast"]

for feature in to_empty_list:
    data[feature] = data[feature].apply(lambda d: d if isinstance(d, list) else [])

to_zero=["Keywords_len", "production_companies_len", "production_countries_len", "crew_len", "cast_len",
    "tagline_word_count", "overview_word_count", "title_word_count"]

for feat in to_zero:
    data[feat]=data[feat].fillna(0)

data['_budget_popularity_ratio'] = data['budget']/data['popularity']
data['_releaseYear_popularity_ratio'] = data['release_year']/data['popularity']
data['_releaseYear_popularity_ratio2'] = data['popularity']/data['release_year']

```

Realice otro .info para ver cómo estaba el data set y note que aun me quedaban unos cuantos atributos no numéricos así que los transforme con la función dummy para poderlos utilizar, asigne ciertos límites para que mi data set siguiera siendo práctico y óptimo.

```

data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 55 columns):
belongs_to_collection      3000 non-null int64
budget                      3000 non-null int64
homepage                    3000 non-null int64
original_language           3000 non-null object
popularity                  3000 non-null float64
production_companies        3000 non-null object
production_countries         3000 non-null object
runtime                     3000 non-null float64
Keywords                    3000 non-null object
cast                        3000 non-null object
revenue                     3000 non-null int64
genre_Action                3000 non-null int64
genre_Adventure              3000 non-null int64
genre_Animation              3000 non-null int64
genre_Comedy                 3000 non-null int64
genre_Crime                  3000 non-null int64
genre_Documentary            3000 non-null int64
genre_Drama                  3000 non-null int64
genre_Family                 3000 non-null int64
genre_Fantasy                 3000 non-null int64
genre_Foreign                 3000 non-null int64
genre_History                 3000 non-null int64
genre_Horror                  3000 non-null int64
genre_Music                  3000 non-null int64
genre_Mystery                 3000 non-null int64
genre_Romance                 3000 non-null int64
genre_Science Fiction          3000 non-null int64
genre_TV Movie                 3000 non-null int64
genre_Thriller                 3000 non-null int64
genre_War                     3000 non-null int64
genre_Western                 3000 non-null int64

```

```

to_dummy = ["original_language", "production_companies", "production_countries", \
            "Keywords", "cast", "Director", "Producer", "Executive Producer"]

limits=[10,35,15,40,30,15,15,20]

for i,feat in enumerate(to_dummy):
    mlb = MultiLabelBinarizer()
    s=data[feat]
    x=pd.DataFrame(mlb.fit_transform(s),columns=mlb.classes_, index=data.index)
    y=pd.DataFrame(mlb.fit_transform(s),columns=mlb.classes_, index=data.index).sum().sort_values(ascending=False)
    rare_entries=y[y<limits[i]].index
    x=x.drop(rare_entries,axis=1)
    data=data.drop(feat,axis=1)
    data=pd.concat([data, x], axis=1, sort=False)

```

Podemos ver que ahora contamos con 140 columnas de tipo float e int, lo que significa que ya todas son numéricas.

```

: data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Columns: 140 entries, belongs_to_collection to Harvey Weinstein
dtypes: float64(12), int64(128)
memory usage: 3.2 MB

```

Me puse a analizar los atributos que eran numéricos desde el principio y note que sí cuentan con outliers así que los elimine, para esto use logaritmos ya que revenue y budget manejaban números muy grandes mientras los demás no.

```

data.revenue = np.log1p(data.revenue)
data.budget = np.log1p(data.budget)

```

```

data.budget.describe()

```

count	3000.000000
mean	11.878667
std	7.441843
min	0.000000
25%	0.000000
50%	15.894952
75%	17.182806
max	19.755682
Name:	budget, dtype: float64

```

data.popularity.describe()

```

count	3000.000000
mean	8.463274
std	12.104000
min	0.000001
25%	4.018053
50%	7.374861
75%	10.890983
max	294.337037
Name:	popularity, dtype: float64

```

: data.runtime.describe()

```

count	3000.000000
mean	107.856571
std	22.079069
min	0.000000
25%	94.000000
50%	104.000000
75%	118.000000
max	338.000000
Name:	runtime, dtype: float64

```

: data.revenue.describe()

```

count	3000.000000
mean	15.961986
std	3.061311
min	0.693147
25%	14.682517
50%	16.637310
75%	18.048445
max	21.141685
Name:	revenue, dtype: float64

Creé otro data set con algunos datos filtrados y los conté para decidir desde donde eliminaba registros y grafiqué para comprobar.

```
data2= data[data[ "runtime"]> 200]
```

```
len(data2)
```

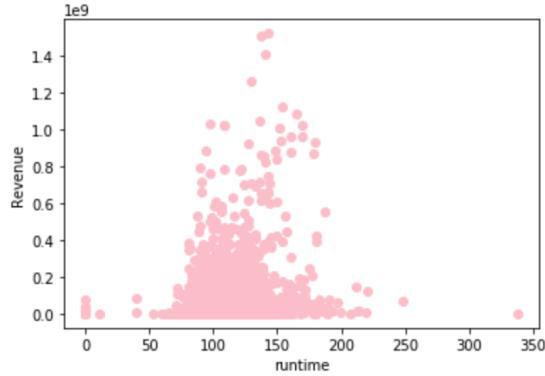
8

```
data2= data[data[ "runtime"]< 60]
```

```
len(data2)
```

16

```
plt.scatter(x=data[ 'runtime'], y=target, color='pink')
plt.ylabel('Revenue')
plt.xlabel('runtime')
plt.show()
```



```
data2= data[data[ "popularity"]<1]
```

```
len(data2)
```

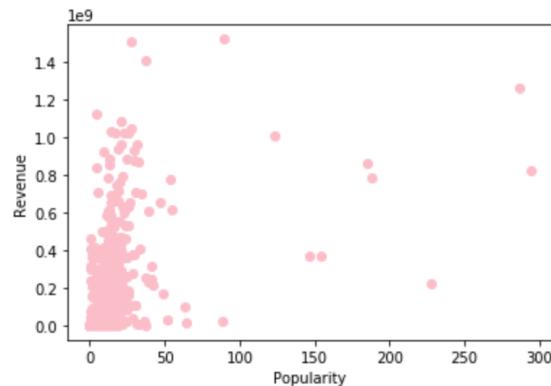
235

```
data2= data[data[ "popularity"]> 50]
```

```
len(data2)
```

15

```
plt.scatter(x=data[ 'popularity'], y=target, color='pink')
plt.ylabel('Revenue')
plt.xlabel('Popularity')
plt.show()
```



```
data2= data[data[ "revenue"]> 19.5]
```

```
len(data2)
```

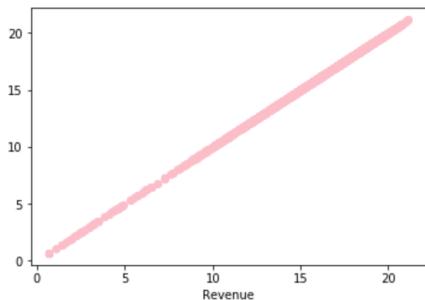
157

```
data2= data[data[ "revenue"]< 4]
```

```
len(data2)
```

33

```
plt.scatter(x=data[ 'revenue'],y=data[ 'revenue'], color='pink')
plt.xlabel('Revenue')
plt.show()
```



```
data2= data[data[ "budget"]== 0]
```

```
len(data2)
```

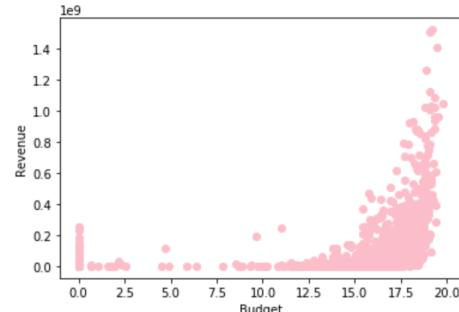
812

```
data2= data[data[ "budget"]> 18]
```

```
len(data2)
```

283

```
plt.scatter(x=data[ 'budget'], y=target, color='pink')
plt.ylabel('Revenue')
plt.xlabel('Budget')
plt.show()
```



Al decidir los límites, elimine los outliers y cree un nuevo data set con todas las características numéricas, saque la correlación entre cada una de ellas y la clase y reduje la dimensionalidad del data set eligiendo las columnas con mayor relación a la clase.

```
data = data[data['runtime'] >=(60)]
data = data[data['runtime'] <=(200)]
```

```
data = data[data['popularity'] >=(1)]
data = data[data['popularity'] <=(50)]
```

```
data = data[data['revenue'] >=(4)]
data = data[data['revenue'] <=(19.5)]
```

```
data = data[data['budget'] <=(18)]
```

```
numeric_features = data.select_dtypes(include=[np.number])
print(numeric_features.dtypes)
```

```
corr = numeric_features.corr()

print (corr['revenue'].sort_values(ascending=False)[:10], '\n')
```

(revenue	1.000000
budget	0.439614
_releaseYear_popularity_ratio2	0.391727
popularity	0.389601
_budget_popularity_ratio	0.303693
has_tagline	0.268498
crew_len	0.248391
United States of America	0.242147
cast_len	0.235792
belongs_to_collection	0.207474
Name: revenue, dtype: float64, '\n'	

Cree el data set con el que voy a entrenar al modelo y dividi mis datos para training y testing.

```
: desired=["budget", "_releaseYear_popularity_ratio2", "popularity", "_budget_popularity_ratio", "has_tagline", "crew_len", '']
: subset = data[desired]
: target=data[ "revenue"]
: 
: Y = target
: X = subset
: 
: X_train, X_test, Y_train, Y_test = train_test_split(X, Y, random_state=42, test_size=.30)
```

Cree los modelos con condiciones arbitrarias, los corrí tres veces con las mismas condiciones para evaluar el cambio del error

Primera:

```
modelos = [
    MLPRegressor(hidden_layer_sizes=30, learning_rate_init=0.0002),
    DecisionTreeRegressor(max_depth=7, max_features=1),
    BaggingRegressor(),
    AdaBoostRegressor(n_estimators=10, learning_rate=0.2),
    RandomForestRegressor(max_depth=7, max_features=1),
    linear_model.LinearRegression()
]

for modelo in modelos:
    modelo.fit(X_train, Y_train)
    # score = classifier.score(X_test, Y_test)
    # print(score)
    rmse = mean_squared_error(Y_test, modelo.predict(X_test))
    print("RMSE:")
    print(rmse)

RMSE:
209.0952529732078
RMSE:
4.663938189089222
RMSE:
4.158977012344437
RMSE:
3.994895121688709
RMSE:
3.8560069114594095
RMSE:
3.840731464479748
```

```
modelos = [
    MLPRegressor(hidden_layer_sizes=30, learning_rate_init=0.0002),
    DecisionTreeRegressor(max_depth=7, max_features=1),
    BaggingRegressor(),
    AdaBoostRegressor(n_estimators=10, learning_rate=0.2),
    RandomForestRegressor(max_depth=7, max_features=1),
    linear_model.LinearRegression()
]

for modelo in modelos:
    modelo.fit(X_train, Y_train)
    # score = classifier.score(X_test, Y_test)
    # print(score)
    rmse = mean_squared_error(Y_test, modelo.predict(X_test))
    print("RMSE:")
    print(rmse)

RMSE:
155.02991252865968
RMSE:
4.60038902487023
RMSE:
4.626470861072452
RMSE:
3.9259054616529947
RMSE:
3.901685310038273
RMSE:
3.840731464479748
```

```
modelos = [
    MLPRegressor(hidden_layer_sizes=30, learning_rate_init=0.0002),
    DecisionTreeRegressor(max_depth=7, max_features=1),
    BaggingRegressor(),
    AdaBoostRegressor(n_estimators=10, learning_rate=0.2),
    RandomForestRegressor(max_depth=7, max_features=1),
    linear_model.LinearRegression()
]

for modelo in modelos:
    modelo.fit(X_train, Y_train)
    # score = classifier.score(X_test, Y_test)
    # print(score)
    rmse = mean_squared_error(Y_test, modelo.predict(X_test))
    print("RMSE:")
    print(rmse)

RMSE:
76.07872269776577
RMSE:
4.62739582393768
RMSE:
4.47993617460299
RMSE:
3.9798270071935002
RMSE:
3.852581363662984
RMSE:
3.840731464479748
```

Podemos observar que en el de red neuronal, el error fluctua mucho pero en los demás es bastante consistente aunque sí varía un poco.

Segundo:

```

modelos = [
    MLPRegressor(hidden_layer_sizes=50, learning_rate_init=0.0002),
    DecisionTreeRegressor(max_depth=10, max_features=1),
    BaggingRegressor(),
    AdaBoostRegressor(n_estimators=15, learning_rate=0.5),
    RandomForestRegressor(max_depth=8, max_features=1),
    linear_model.LinearRegression()
]

for modelo in modelos:
    modelo.fit(X_train, Y_train)
    # score = classifier.score(X_test, Y_test)
    # print(score)
    rmse = mean_squared_error(Y_test, modelo.predict(X_test))
    print("RMSE:")
    print(rmse)

RMSE:
56.35241079017002
RMSE:
5.1529484830115
RMSE:
4.459861523359038
RMSE:
4.3652839581037295
RMSE:
3.9384342441497786
RMSE:
3.840731464479748

```



```

modelos = [
    MLPRegressor(hidden_layer_sizes=50, learning_rate_init=0.0002),
    DecisionTreeRegressor(max_depth=10, max_features=1),
    BaggingRegressor(),
    AdaBoostRegressor(n_estimators=15, learning_rate=0.5),
    RandomForestRegressor(max_depth=8, max_features=1),
    linear_model.LinearRegression()
]

for modelo in modelos:
    modelo.fit(X_train, Y_train)
    # score = classifier.score(X_test, Y_test)
    # print(score)
    rmse = mean_squared_error(Y_test, modelo.predict(X_test))
    print("RMSE:")
    print(rmse)

RMSE:
246.94681262516767
RMSE:
5.479568706365284
RMSE:
4.491651010847585
RMSE:
4.267787164499277
RMSE:
3.9787797348307707
RMSE:
3.840731464479748

```



```

modelos = [
    MLPRegressor(hidden_layer_sizes=50, learning_rate_init=0.0002),
    DecisionTreeRegressor(max_depth=10, max_features=1),
    BaggingRegressor(),
    AdaBoostRegressor(n_estimators=15, learning_rate=0.5),
    RandomForestRegressor(max_depth=8, max_features=1),
    linear_model.LinearRegression()
]

for modelo in modelos:
    modelo.fit(X_train, Y_train)
    # score = classifier.score(X_test, Y_test)
    # print(score)
    rmse = mean_squared_error(Y_test, modelo.predict(X_test))
    print("RMSE:")
    print(rmse)

RMSE:
3307.038653584474
RMSE:
7.071930510970519
RMSE:
4.490134232830236
RMSE:
4.348117125049928
RMSE:
3.970307448974179
RMSE:
3.840731464479748

```

Aqua decidí a los árboles darles un poco más de profundidad pero dado los resultados creo que fue una mejor combinación la primera. También aumente el tamaño del MLPREGRESSOR y consideró que fue contraproducente ya que lo hizo más fluctuante y no más preciso.

Tercero:

```

modelos = [
    MLPRegressor(hidden_layer_sizes=40, learning_rate_init=0.0002),
    DecisionTreeRegressor(max_depth=7, max_features=3),
    BaggingRegressor(),
    AdaBoostRegressor(n_estimators=8, learning_rate=0.2),
    RandomForestRegressor(max_depth=8, max_features=2),
    linear_model.LinearRegression()
]

for modelo in modelos:
    modelo.fit(X_train, Y_train)
    # score = classifier.score(X_test, Y_test)
    # print(score)
    rmse = mean_squared_error(Y_test, modelo.predict(X_test))
    print("RMSE:")
    print(rmse)

```

```

RMSE:
364.7720972760944
RMSE:
4.479927662821252
RMSE:
4.307572244359194
RMSE:
3.9612268113843583
RMSE:
3.8162853644076713
RMSE:
3.840731464479748

```

```

modelos = [
    MLPRegressor(hidden_layer_sizes=40, learning_rate_init=0.0002),
    DecisionTreeRegressor(max_depth=7, max_features=3),
    BaggingRegressor(),
    AdaBoostRegressor(n_estimators=8, learning_rate=0.2),
    RandomForestRegressor(max_depth=8, max_features=2),
    linear_model.LinearRegression()
]

for modelo in modelos:
    modelo.fit(X_train, Y_train)
    # score = classifier.score(X_test, Y_test)
    # print(score)
    rmse = mean_squared_error(Y_test, modelo.predict(X_test))
    print("RMSE:")
    print(rmse)

```

```

RMSE:
244.79899758848237
RMSE:
4.4025727402914265
RMSE:
4.500398784783504
RMSE:
4.0188194571426274
RMSE:
3.7157152488612106
RMSE:
3.840731464479748

```

```

modelos = [
    MLPRegressor(hidden_layer_sizes=40, learning_rate_init=0.0002),
    DecisionTreeRegressor(max_depth=7, max_features=3),
    BaggingRegressor(),
    AdaBoostRegressor(n_estimators=8, learning_rate=0.2),
    RandomForestRegressor(max_depth=8, max_features=2),
    linear_model.LinearRegression()
]

for modelo in modelos:
    modelo.fit(X_train, Y_train)
    # score = classifier.score(X_test, Y_test)
    # print(score)
    rmse = mean_squared_error(Y_test, modelo.predict(X_test))
    print("RMSE:")
    print(rmse)

```

```

RMSE:
18.803940598371927
RMSE:
4.305363025571859
RMSE:
4.506287365595013
RMSE:
3.971943277215782
RMSE:
3.940928977915409
RMSE:
3.840731464479748

```

Apesar de que los valores se eligieron arbitrariamente considero que la mejor combinación fue la primera ya que es más estable, menos variante y más optima.

CONCLUSIÓN

Con este proyecto entendí que hay muchas maneras de analizar, normalizar, limpiar, y transformar datos, que aun que no puedas hacerlo de una forma , puedes lograrlo de otra. Con respecto a lo modelos me di cuenta que pequeños cambios hacen una gran diferencia y que en el caso de este tema mi primer análisis fue el más exitoso.

Me di cuenta que de mis hipótesis solo tres atributos eran correctos siendo el de Popularity, Cast y Crew y descubrí que a pesar de mis creencias de que no se necesariamente por tener un presupuesto grande va a ganar mucho si lleva una relación directa con las ganancias de la película.