

# Projeto final de Machine Learning

## Previsão de receita de filmes

Alunos: Gabriela Caruso, Guilherme Roccato e Pedro Paulo Telho

## Introdução

A indústria cinematográfica é reconhecida por suas diversas obras primas. Prêmios como Oscar, Globo de Ouro e Emmy são apenas alguns dos que recompensam os seus atores, diretores, produtores etc. Além disso, muito dinheiro é movimentado tanto em forma de investimento para a produção dos filmes como em forma de receita após o lançamento.

Nesse contexto, o objetivo do projeto é prever quanto de receita um filme pode gerar para que não se tenha prejuízo durante suas produções.

Para isso, será criado um modelo de regressão através da base de dados: [TMDb 5000 movies dataset](#)

## Metodologia

Nessa seção será apresentada a forma em que o modelo foi construído: bibliotecas necessárias, exploração dos dados e técnicas de inferência para definição do melhor modelo de machine learning para o problema.

### Importando bibliotecas necessárias:

```
In [1]: %matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

import sys

import numpy as np
import scipy.sparse

import sklearn.linear_model
import sklearn.datasets
import sklearn.svm
import sklearn.metrics
import sklearn.decomposition
import sklearn.feature_extraction.text
import sklearn.utils.sparsefuncs

import logging

import json
```

```
In [2]: RANDOM_SEED = 42
np.random.seed(RANDOM_SEED)
```

### Carregando os dados:

```
In [3]: movies = pd.read_csv("tmdb_5000_movies.csv")
credits = pd.read_csv("tmdb_5000_credits.csv")
```

Mesclando créditos e filmes:

```
In [4]: credits = credits[['movie_id', 'cast', 'crew']]
credits = credits.rename(columns = {'movie_id': 'id'})
movies = pd.merge(movies, credits, on='id', how='left')
```

```
In [5]: movies_copy = movies.copy()
```

```
In [6]: movies_copy.head()
```

										Out[6]:
	budget	genres	homepage	id	keywords	original_language	original_title	overview	popularit	
0	237000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.avatarmovie.com/	19995	[{"id": 1463, "name": "culture clash"}, {"id": ...	en	Avatar	In the 22nd century, a paraplegic Marine is di...	150.43757	
1	300000000	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	http://disney.go.com/disneypictures/pirates/	285	[{"id": 270, "name": "ocean"}, {"id": 726, "na...	en	Pirates of the Caribbean: At World's End	Captain Barbosa, long believed to be dead, ha...	139.08261	
2	245000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.sonypictures.com/movies/spectre/	206647	[{"id": 470, "name": "spy"}, {"id": 818, "name...	en	Spectre	A cryptic message from Bond's past sends him o...	107.37678	
3	250000000	[{"id": 28, "name": "Action"}, {"id": 80, "nam...	http://www.thedarkknighttrises.com/	49026	[{"id": 849, "name": "dc comics"}, {"id": 853,...	en	The Dark Knight Rises	Following the death of District Attorney Harve...	112.31295	
4	260000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://movies.disney.com/john-carter	49529	[{"id": 818, "name": "based on novel"}, {"id": ...	en	John Carter	John Carter is a war-weary, former military ca...	43.92699	

5 rows x 22 columns

Selecionando apenas filmes lançados:

```
In [7]: movies_copy = movies_copy.drop(movies_copy[movies_copy.status != 'Released'].index)
```

Deletando features que não são interessantes para a análise:

```
In [8]: movies_copy = movies_copy.drop(columns=['homepage', 'id', 'original_title', 'tagline', 'status', 'cre
```

## Exploração de dados

```
In [9]: movies_copy.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4795 entries, 0 to 4802
Data columns (total 16 columns):
budget                4795 non-null int64
genres                4795 non-null object
keywords              4795 non-null object
original_language     4795 non-null object
overview              4792 non-null object
popularity            4795 non-null float64
production_companies  4795 non-null object
production_countries  4795 non-null object
release_date          4794 non-null object
revenue               4795 non-null int64
runtime               4793 non-null float64
spoken_languages      4795 non-null object
title                 4795 non-null object
vote_average          4795 non-null float64
vote_count            4795 non-null int64
cast                  4795 non-null object
dtypes: float64(3), int64(3), object(10)
memory usage: 636.8+ KB
```

Deletando linhas com valores nulos ou zerados:

```
In [10]: movies_copy = movies_copy.dropna()
```

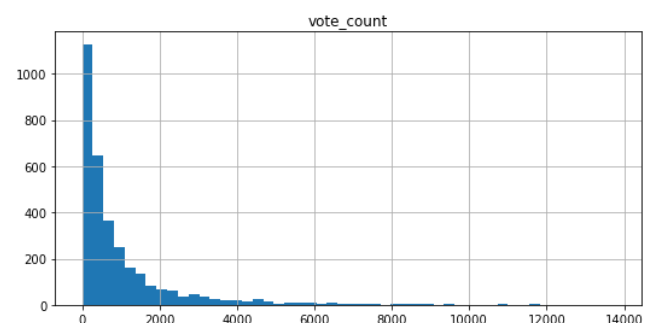
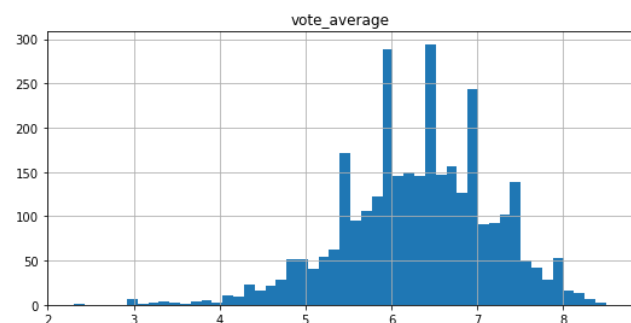
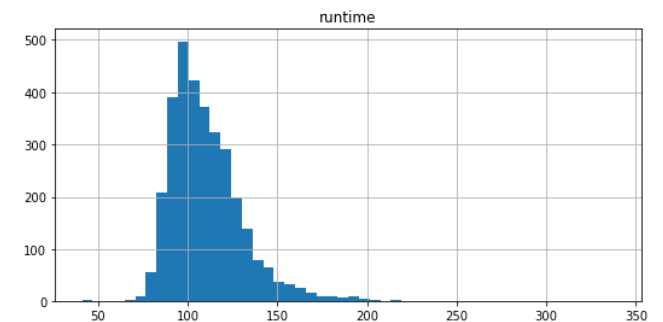
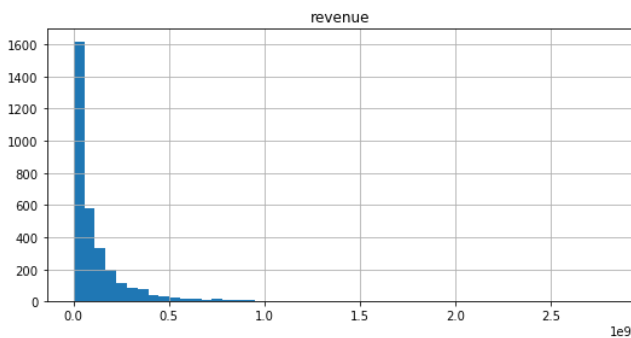
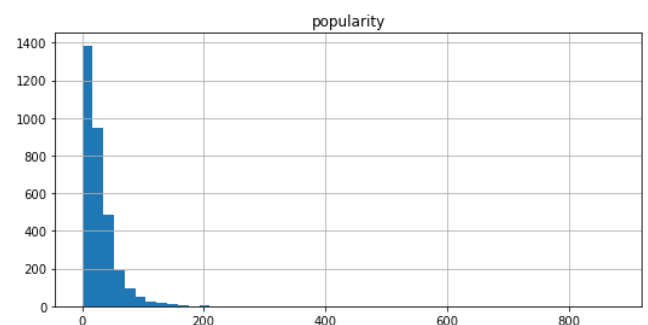
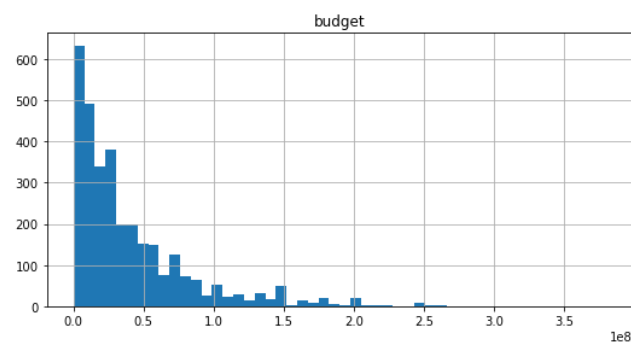
```
In [11]: movies_copy = movies_copy[(movies_copy[['budget', 'revenue', 'runtime', 'vote_count']] != 0).all(axis=1)]
```

## Dataset limpo:

```
In [12]: movies_copy.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3226 entries, 0 to 4798
Data columns (total 16 columns):
budget                3226 non-null int64
genres                3226 non-null object
keywords              3226 non-null object
original_language     3226 non-null object
overview              3226 non-null object
popularity             3226 non-null float64
production_companies  3226 non-null object
production_countries  3226 non-null object
release_date          3226 non-null object
revenue               3226 non-null int64
runtime               3226 non-null float64
spoken_languages      3226 non-null object
title                 3226 non-null object
vote_average          3226 non-null float64
vote_count            3226 non-null int64
cast                  3226 non-null object
dtypes: float64(3), int64(3), object(10)
memory usage: 428.5+ KB
```

```
In [13]: movies_copy.hist(bins=50, figsize=(20, 15))
plt.show()
```



Visualizando os histogramas acima é possível perceber que não há nenhuma anomalia com a base de dados e, portanto, é possível continuar com a análise sem nenhuma necessidade de manipulação.

```
In [14]: correlation_matrix = movies_copy.corr()
```

Com a matriz de correlação abaixo pode-se inferir também que a receita de cada filme tem maior correlação com a quantidade de votos.

```
In [15]: correlation_matrix
```

Out[15]:

	budget	popularity	revenue	runtime	vote_average	vote_count
budget	1.000000	0.431629	0.705260	0.229717	-0.036087	0.539916
popularity	0.431629	1.000000	0.602063	0.182385	0.288001	0.748965
revenue	0.705260	0.602063	1.000000	0.233235	0.187830	0.756110
runtime	0.229717	0.182385	0.233235	1.000000	0.386253	0.258103
vote_average	-0.036087	0.288001	0.187830	0.386253	1.000000	0.380685
vote_count	0.539916	0.748965	0.756110	0.258103	0.380685	1.000000

Uma possível razão para tal é que quanto maior a receita gerada por um filme maior alcance esse tem. Assim, uma maior quantidade de pessoas pode fazer parte da votação. Por isso, essa correlação é forte. A mesma lógica pode ser aplicada quando a quantidade de votos é analisada em relação à popularidade.

### Análise de texto:

As features que não são quantitativas são representadas por objetos texto. Nesse sentido, é necessário adaptar o modelo para continuar a análise de modelos de regressão.

Para analisar textos foi utilizada a função do scikit-learn **CountVectorizer** com o intuito de transformar textos em vetores numéricos e a função **TruncatedSVD** para melhorar a eficiência de trabalho com matrizes esparsas.

```
In [16]: movies_ = movies_copy
```

```
In [17]: movies_copy = movies_copy.drop(columns = {'title','release_date','production_countries','spoken_lang
```

```
In [18]: text_key = ['genres', 'keywords', 'production_companies', 'cast']
```

```
for j in text_key:
    movies_copy[j] = movies_copy[j].apply(lambda x:[i['name'] for i in json.loads(str(x))])
#     movies_copy[j] = movies_copy[j].apply(lambda x:", ".join(x))
```

```
In [19]: movies_copy.keywords
```

Out[19]:

```
0      [culture clash, future, space war, space colon...
1      [ocean, drug abuse, exotic island, east india ...
2      [spy, based on novel, secret agent, sequel, mi...
3      [dc comics, crime fighter, terrorist, secret i...
4      [based on novel, mars, medallion, space travel...
...
4773      [salesclerk, loser, aftercreditsstinger]
4788      [gay, trailer park, pop culture, drug dealer, ...
4792      [japan, prostitute, hotel, based on novel, hal...
4796      [distrust, garage, identity crisis, time trave...
4798      [united states-mexico barrier, legs, arms, pap...
Name: keywords, Length: 3226, dtype: object
```

Separando o tamanho de componentes de cada feature. A estratégia é captar 50% do valor da variância de cada coluna. Somente para as colunas "keywords", "genres", "original\_language", "production\_companies" e "cast" a estratégia irá funcionar. Para os valores de "overview" obter-se-a através de tentativa e erro.

```
In [20]: text_to_num = ['keywords','genres', 'original_language', 'production_companies', 'cast', 'overview']
```

```
list_n = []
for i in text_to_num:
    if i != 'overview':
        n_elem = []
        movies_copy[i].apply(lambda x: n_elem.append(len(x)))
        list_n.append(int(np.round(np.var(n_elem)*0.5)))
print(list_n)
```

```
[20, 1, 0, 2, 231]
```

```
In [21]: list_n.append(50)
```

```
In [22]: list_of_dfs = []
```

```
for i, n in zip(text_to_num, list_n):
    corpus = movies_copy[i]

    vectorizer = sklearn.feature_extraction.text.CountVectorizer(
        ngram_range=(1, 1), #ngram_range=(1, 1) is the default
        dtype='double',
        analyzer=lambda x: x
    )
    data = vectorizer.fit_transform(corpus)

    pca = sklearn.decomposition.TruncatedSVD(n_components=n)
    data = pca.fit_transform(data)

    column_names = []
    for j in range(len(data.tolist()[0])):
        col_name = i + "_" + str(j)
        column_names.append(col_name)
    new_data = pd.DataFrame(data.tolist(), columns=column_names)

    list_of_dfs.append(new_data)
```

```
In [23]: list_of_dfs.insert(0, movies_copy)
movies_copy = pd.concat(list_of_dfs, axis=1)
movies_copy.head()
```

Out[23]:

	budget	genres	keywords	original_language	overview	popularity	production_companies	revenue	runtime	vote_average
0	237000000.0	[Action, Adventure, Fantasy, Science Fiction]	[culture clash, future, space war, space colon...]	en	In the 22nd century, a paraplegic Marine is di...	150.437577	[Ingenious Film Partners, Twentieth Century Fo...	2.787965e+09	162.0	7.2
1	300000000.0	[Adventure, Fantasy, Action]	[ocean, drug abuse, exotic island, east india ...]	en	Captain Barbosa, long believed to be dead, ha...	139.082615	[Walt Disney Pictures, Jerry Bruckheimer Films...	9.610000e+08	169.0	6.9
2	245000000.0	[Action, Adventure, Crime]	[spy, based on novel, secret agent, sequel, mi...]	en	A cryptic message from Bond's past sends him o...	107.376788	[Columbia Pictures, Danjaq, B24]	8.806746e+08	148.0	6.3
3	250000000.0	[Action, Crime, Drama, Thriller]	[dc comics, crime fighter, terrorist, secret i...]	en	Following the death of District Attorney Harve...	112.312950	[Legendary Pictures, Warner Bros., DC Entertai...	1.084939e+09	165.0	7.6
4	260000000.0	[Action, Adventure, Science Fiction]	[based on novel, mars, medallion, space travel...]	en	John Carter is a war-weary, former military ca...	43.926995	[Walt Disney Pictures]	2.841391e+08	132.0	6.1

5 rows x 316 columns

```
In [24]: movies_copy = movies_copy.drop(columns={'genres', 'keywords', 'original_language', 'production_compa
```

```
In [25]: movies_copy.head()
```

Out[25]:

	budget	popularity	revenue	runtime	vote_average	vote_count	keywords_0	keywords_1	keywords_2	keywords_3	...	overview
0	237000000.0	150.437577	2.787965e+09	162.0	7.2	11800.0	0.371568	0.006018	-0.197439	-0.504144	...	0.258
1	300000000.0	139.082615	9.610000e+08	169.0	6.9	4500.0	0.394198	-0.254569	0.097741	-0.004706	...	-0.520
2	245000000.0	107.376788	8.806746e+08	148.0	6.3	4466.0	0.427153	0.263526	-0.493112	-0.570878	...	-0.837
3	250000000.0	112.312950	1.084939e+09	165.0	7.6	9106.0	0.302618	0.022682	0.288550	-0.242756	...	1.060
4	260000000.0	43.926995	2.841391e+08	132.0	6.1	2124.0	0.545083	0.199346	-0.659397	-0.801513	...	0.469

5 rows × 310 columns

```
In [26]: movies_copy = movies_copy.dropna()
```

Com a alteração realizada, é possível analisar novamente os valores de correlação dos dados. A feature **vote\_count** continua sendo a mais importante para a regressão.

```
In [27]: correlation_matrix = movies_copy.corr()
correlation_matrix.unstack().dropna().sort_values(ascending=False).drop_duplicates()
```

Out[27]:

```
overview_49  overview_49    1.000000
cast_31      cast_22      0.925254
revenue      vote_count    0.760050
vote_count   popularity    0.732853
revenue      budget       0.685696
...
keywords_3    budget     -0.308860
cast_55       cast_67     -0.380296
cast_14       cast_16     -0.447061
cast_18       cast_15     -0.473399
cast_56       cast_67     -0.596228
Length: 47896, dtype: float64
```

## Separação de dados treinamento/teste

Para manter a integridade dos dados de treinamento/teste é necessário fazer uma separação estratificada. Nesse sentido, foi realizada a separação pela variável **vote\_count**.

```
In [28]: import math
ref = movies_copy['vote_count'].max()
min_ref = movies_copy['vote_count'].min()
first = math.ceil((ref-min_ref) / 5)
second = math.ceil((ref-min_ref) / 5*2)
third = math.ceil((ref-min_ref) / 5*3)
fourth = math.ceil((ref-min_ref) / 5*4)
```

```
In [29]: def stratified_shuffle(x):
    if x <= first:
        return 1.0
    elif x > first and x <= second:
        return 2.0
    elif x > second and x <= third:
        return 3.0
    elif x > third and x <= fourth:
        return 4.0
    else:
        return 5.0
```

```
In [30]: movies_copy['vote_count_cat'] = movies_copy['vote_count'].astype(int)
```

```
In [31]: movies_copy['vote_count_cat'] = movies_copy['vote_count'].apply(stratified_shuffle)
```

```
In [32]: movies_copy['vote_count_cat'].value_counts(True)
```

Out[32]:

```
1.0    0.892607
2.0    0.080934
3.0    0.019455
4.0    0.005447
5.0    0.001556
Name: vote_count_cat, dtype: float64
In [33]: movies_copy.head()
```

Out[33]:

	budget	popularity	revenue	runtime	vote_average	vote_count	keywords_0	keywords_1	keywords_2	keywords_3	...	overview
0	237000000.0	150.437577	2.787965e+09	162.0	7.2	11800.0	0.371568	0.006018	-0.197439	-0.504144	...	0.096
1	300000000.0	139.082615	9.610000e+08	169.0	6.9	4500.0	0.394198	-0.254569	0.097741	-0.004706	...	0.762
2	245000000.0	107.376788	8.806746e+08	148.0	6.3	4466.0	0.427153	0.263526	-0.493112	-0.570878	...	-0.441
3	250000000.0	112.312950	1.084939e+09	165.0	7.6	9106.0	0.302618	0.022682	0.288550	-0.242756	...	-0.670
4	260000000.0	43.926995	2.841391e+08	132.0	6.1	2124.0	0.545083	0.199346	-0.659397	-0.801513	...	0.642

5 rows × 311 columns

```
In [34]: movies_copy = movies_copy.reset_index()

In [35]: movies_copy = movies_copy.drop(columns = {'index'})

Divisão do conjunto de dados de modo estratificado:

In [36]: from sklearn.model_selection import StratifiedShuffleSplit

        split = StratifiedShuffleSplit(
            n_splits=1,
            test_size=0.2,
            random_state=RANDOM_SEED,
        )
        for train_index, test_index in split.split(movies_copy, movies_copy['vote_count_cat']):
            strat_train_set = movies_copy.loc[train_index]
            strat_test_set = movies_copy.loc[test_index]
```

Verificando se a separação estratificada foi bem sucedida:

```
In [37]: strat_train_set['vote_count_cat'].value_counts(True)
```

Out[37]:

```
1.0    0.892510
2.0    0.081226
3.0    0.019455
4.0    0.005350
5.0    0.001459
Name: vote_count_cat, dtype: float64
In [38]: strat_test_set['vote_count_cat'].value_counts(True)
```

Out[38]:

```
1.0    0.892996
2.0    0.079767
3.0    0.019455
4.0    0.005837
5.0    0.001946
Name: vote_count_cat, dtype: float64
```

Removendo a nova coluna que foi adicionada temporariamente:

```
In [39]: strat_train_set.drop(['vote_count_cat'], axis=1, inplace=True)
        strat_test_set.drop(['vote_count_cat'], axis=1, inplace=True)
```

A partir de agora tem-se os conjuntos de treinamento e teste, **strat\_train\_set** e **strat\_test\_set**, respectivamente. O conjunto **strat\_train\_set** será usado para várias explorações e para construir nosso modelo preditivo. Já o conjunto **strat\_test\_set** será usado só no final do projeto, para avaliar a performance final do modelo.

## Preparando os dados para o modelo - Separando X e y

Com os conjuntos separados, é possível iniciar a construção de um modelo de machine learning para o problema.

Como o **dataset** foi "limpado" e criou-se referências em números para os objetos textos, basta separar as variáveis e construir os modelos preditivos.

```
In [40]: strat_train_set_copy = strat_train_set.copy()
        strat_test_set_copy = strat_test_set.copy()
```

Variáveis independentes (dataset original menos a coluna de valores dependentes):

```
In [41]: movies = strat_train_set_copy.drop('revenue', axis=1)
```

Variável dependente (também chamada de label):

```
In [42]: movies_labels = strat_train_set_copy['revenue'].copy()
```

## Construindo modelos preditivos

### REGRESSÃO LINEAR:

O modelo mais simples de regressão é a **regressão linear**, na qual deseja-se estimar um valor dependente como uma combinação linear dos valores independentes. Para esse fim, é necessário treinar um modelo no Scikit-Learn a partir da criação de um regressor, e da chamada do método **fit()** deste regressor para ajustar os parâmetros internos do modelo:

```
In [43]: from sklearn.linear_model import LinearRegression

        lin_reg = LinearRegression()
        lin_reg.fit(movies, movies_labels)
```

Out[43]:

```
LinearRegression()
```

Para obter as previsões, basta chamar o método `predict()` e comparar com os valores originais para verificar o modelo:

```
In [44]: # previsões
        predicted_labels = lin_reg.predict(movies)
        print('Predição: {}'.format(predicted_labels.round(decimals=2)))

        # valores originais:
        print('Original: {}'.format(movies_labels.values.round(decimals=2)))
```

```
Predição: [7.45034540e+07 5.06566968e+07 8.73136573e+07 ... 4.52847063e+08
 1.25031954e+08 5.75524319e+08]
```

```
Original: [8.54985340e+07 1.20361490e+07 5.49561400e+07 ... 3.49424282e+08
 1.11811453e+08 9.40335536e+08]
```

Para verificar a eficácia do modelo, pode-se medir o erro de predição no conjunto de treinamento:

```
In [45]: from sklearn.metrics import mean_squared_error

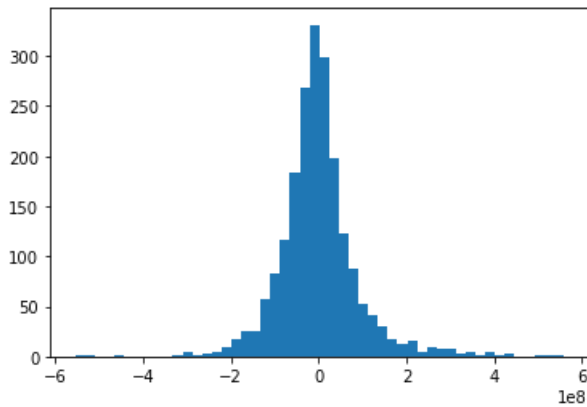
        movies_predictions = lin_reg.predict(movies)
        lin_mse = mean_squared_error(movies_labels, movies_predictions)
        lin_rmse = np.sqrt(lin_mse)
        print('Regressão linear: RMSE = {:.2f}'.format(lin_rmse))
```

```
Regressão linear: RMSE = 92179758.78
```

```
In [46]: residuo = movies_labels - movies_predictions
```

```
In [47]: plt.hist(residuo, bins=50);
```





```
In [48]: pd.Series(residuo).describe()
```

```
count    2.056000e+03
mean     -2.458402e-08
std       9.220218e+07
min      -5.546957e+08
25%      -4.591203e+07
50%      -5.362647e+06
75%       3.450751e+07
max       5.576419e+08
Name: revenue, dtype: float64
```

Out[48]:

### ÁRVORE DE DECISÃO:

```
In [49]: from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor(random_state=RANDOM_SEED)
tree_reg.fit(movies, movies_labels)
```

Out[49]:

```
DecisionTreeRegressor(random_state=42)
```

```
In [50]: predicted_labels = tree_reg.predict(movies)
print('Predição: {}'.format(predicted_labels))
print('Original: {}'.format(movies_labels.values))
```

```
Predição: [8.54985340e+07 1.20361490e+07 5.49561400e+07 ... 3.49424282e+08
1.11811453e+08 9.40335536e+08]
Original: [8.54985340e+07 1.20361490e+07 5.49561400e+07 ... 3.49424282e+08
1.11811453e+08 9.40335536e+08]
```

```
In [51]: movies_predictions = tree_reg.predict(movies)
tree_mse = mean_squared_error(movies_labels, movies_predictions)
tree_rmse = np.sqrt(tree_mse)
print('Árvore de decisão: RMSE = {:.2f}'.format(tree_rmse))
```

```
Árvore de decisão: RMSE = 0.00
```

### Overfitting!

### RANDOM FOREST:

```
In [52]: from sklearn.ensemble import RandomForestRegressor

forest_reg = RandomForestRegressor(n_estimators=10, random_state=RANDOM_SEED)
```

```
In [53]: forest_reg.fit(movies, movies_labels)

y_pred = forest_reg.predict(movies)
forest_rmse = np.sqrt(mean_squared_error(movies_labels, movies_predictions))
print('Regressão random forest: RMSE = {:.2f}'.format(forest_rmse))
```

```
Regressão random forest: RMSE = 0.00
```

### Overfitting!

## Melhorando a avaliação usando validação cruzada

Para escolher um bom modelo, é necessário tentar estimar o erro de teste do modelo. Nesse sentido, deseja-se treinar o modelo em um certo conjunto de dados, e testá-lo em outro conjunto de dados. Para tal, divide-se o conjunto de treinamento em dois novamente! Usa-se um pedaço para treinar modelos, e outro para testar desempenho.

Isso é necessário pois os verdadeiros dados de teste do modelo não devem ser tocados até que toda a análise esteja pronta.

Por simplicidade a equipe usa a divisão não-estratificada:

```
In [54]: from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    movies,
    movies_labels,
    test_size=0.2,
    random_state=RANDOM_SEED,
)
```

## REGRESSÃO LINEAR:

```
In [55]: lin_reg.fit(X_train, y_train)

y_pred_rl = lin_reg.predict(X_test)
lin_rmse = np.sqrt(mean_squared_error(y_test, y_pred_rl))
print('Regressão linear: RMSE = {:.2f}'.format(lin_rmse))
```

Regressão linear: RMSE = 391451193.39

## ÁRVORE DE DECISÃO:

```
In [56]: tree_reg.fit(X_train, y_train)

y_pred_tr = tree_reg.predict(X_test)
tree_rmse = np.sqrt(mean_squared_error(y_test, y_pred_tr))
print('Árvore de decisão: RMSE = {:.2f}'.format(tree_rmse))
```

Árvore de decisão: RMSE = 129084889.47

## RANDOM FOREST:

```
In [57]: forest_reg.fit(X_train, y_train)

y_pred_rf = forest_reg.predict(X_test)
forest_rmse = np.sqrt(mean_squared_error(y_test, y_pred_rf))
print('Random Forest: RMSE = {:.2f}'.format(forest_rmse))
```

Random Forest: RMSE = 103081091.88

Parece que o random forest é melhor que os outros modelos, mas talvez todos esses resultados sejam pura sorte. Para verificar pode-se repetir esses experimentos com partições diferentes e ver se o resultado se mantém. O scikit-learn já tem ferramentas para ajudar nessa tarefa:

## REGRESSÃO LINEAR:

```
In [58]: from sklearn.model_selection import cross_val_score

lin_scores = cross_val_score(
    lin_reg, # modelo
    movies, # X_train
    movies_labels, # y_train
    scoring='neg_mean_squared_error', # metrica de interesse
    cv=10, # Quantas partições eu quero
    n_jobs=-1, # Use todas as cores da maquina!
)
lin_rmse_scores = np.sqrt(-lin_scores)

def display_scores(scores):
    print('Scores:', scores.round(decimals=2))
    print('Mean:', scores.mean())
    print('Standard deviation:', scores.std())
```

```
display_scores(lin_rmse_scores)
```

```
Scores: [1.37704007e+08 1.84520834e+08 2.04737099e+08 1.92591936e+08
 4.38167946e+08 2.48310124e+08 2.18139882e+08 1.46773156e+08
 3.91404373e+08 1.60299378e+08]
Mean: 232264873.69192296
Standard deviation: 97066605.18578978
```

### ÁRVORE DE DECISÃO:

```
In [59]: tree_scores = cross_val_score(
        tree_reg,
        movies,
        movies_labels,
        scoring='neg_mean_squared_error',
        cv=10,
        n_jobs=-1,
    )
    tree_rmse_scores = np.sqrt(-tree_scores)
    display_scores(tree_rmse_scores)
```

```
Scores: [1.83215566e+08 1.25521282e+08 1.78557888e+08 1.67642065e+08
 1.48125798e+08 1.77079129e+08 1.96654286e+08 1.75255477e+08
 1.60436240e+08 1.51926268e+08]
Mean: 166441399.971425
Standard deviation: 19467818.603153493
```

### RANDOM FOREST:

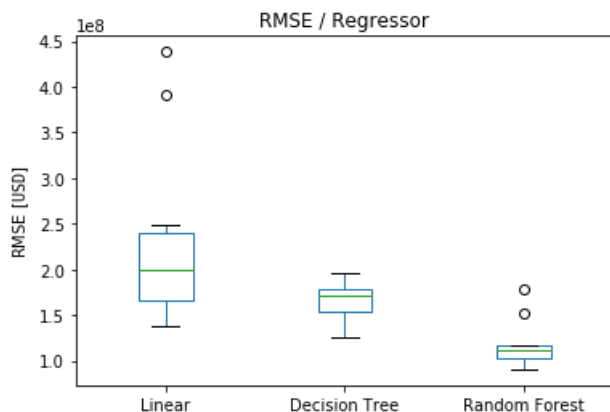
```
In [60]: forest_scores = cross_val_score(
        forest_reg,
        movies,
        movies_labels,
        scoring='neg_mean_squared_error',
        cv=10,
        n_jobs=-1,
    )
    forest_rmse_scores = np.sqrt(-forest_scores)
    display_scores(forest_rmse_scores)
```

```
Scores: [9.20505506e+07 9.06151183e+07 1.02314001e+08 1.17472798e+08
 1.13347116e+08 1.51848752e+08 1.77897337e+08 1.04725678e+08
 1.10238664e+08 1.14064282e+08]
Mean: 117457429.7141074
Standard deviation: 25833514.193084817
```

Comparando os três modelos:

```
In [61]: pd.DataFrame({
        'Linear': lin_rmse_scores,
        'Decision Tree': tree_rmse_scores,
        'Random Forest': forest_rmse_scores,
    }).plot.box();
plt.ylabel(r'RMSE  $\$[\text{USD}]$ ')
plt.title("RMSE / Regressor")
```

```
Text(0.5, 1.0, 'RMSE / Regressor')
```



Agora sim pode-se perceber que o regressor **random forest** é melhor que os outros uma vez que apresenta um menor RMSE.

Out[61]:

```
In [62]: import time

from sklearn.model_selection import GridSearchCV

parameters = {'fit_intercept':[True,False], 'normalize':[True,False], 'copy_X':[True, False]}

lin_reg = LinearRegression()

grid_search = GridSearchCV(
    lin_reg, # Modelo
    parameters, # Grid
    cv=5, # Partições de C.V.
    scoring='neg_mean_squared_error',
    return_train_score=True,
    n_jobs=-1,
)

t1 = time.perf_counter()
grid_search.fit(movies, movies_labels)
t2 = time.perf_counter()

print(f'Tempo gasto: {t2 - t1:.2f} s')
```

Tempo gasto: 0.98 s

```
In [63]: grid_search.best_params_
```

Out[63]:

```
{'copy_X': True, 'fit_intercept': True, 'normalize': True}
```

```
In [64]: grid_search.best_estimator_
```

Out[64]:

```
LinearRegression(normalize=True)
```

```
In [65]: cvres = grid_search.cv_results_
for mean_score, params in zip(cvres['mean_test_score'], cvres['params']):
    print(np.sqrt(-mean_score), params)
```

```
293599025.8270065 {'copy_X': True, 'fit_intercept': True, 'normalize': True}
293599028.3501676 {'copy_X': True, 'fit_intercept': True, 'normalize': False}
295225338.192539 {'copy_X': True, 'fit_intercept': False, 'normalize': True}
295225338.192539 {'copy_X': True, 'fit_intercept': False, 'normalize': False}
293599025.8270065 {'copy_X': False, 'fit_intercept': True, 'normalize': True}
293599028.3501676 {'copy_X': False, 'fit_intercept': True, 'normalize': False}
295225338.192539 {'copy_X': False, 'fit_intercept': False, 'normalize': True}
295225338.192539 {'copy_X': False, 'fit_intercept': False, 'normalize': False}
```

Assim, finalmente é possível medir o desempenho final do modelo:

```
In [66]: final_model = grid_search.best_estimator_

X_test = strat_test_set.drop('revenue', axis=1)
y_test = strat_test_set['revenue'].copy()

final_model.fit(X_test, y_test)

final_predictions = final_model.predict(X_test)

final_mse = mean_squared_error(y_test, final_predictions)
final_rmse = np.sqrt(final_mse)

print(f'RMSE = {final_rmse}')
```

RMSE = 63879425.07142814

## Conclusão

A partir de toda a análise exploratória e construção de diferentes modelos preditivos foi possível identificar o **regressor random forest** como o melhor para solucionar esse problema uma vez que apresenta o menor valor de RMSE e menor desvio padrão.

No entanto, é necessário uma análise bem mais rebuscada para que a precisão de inferência seja boa o suficiente para se colocar em prática.

Porém, pelo escopo desse projeto, foi possível desenvolver um modelo de *Machine Learning* capaz de prever a receita de um filme considerando suas outras características.

In [ ]: