International University of Sarajevo

Faculty of Engineering and Natural Sciences

Introduction to Machine Learning
[EE418]

Project report

Title: Prediction of movies revenues using machine learning

Supervised by: Assist.Prof.Dr. Kanita Karadjuzovic-Hadziabdic
Computer Science and Engineering department, FENS, IUS

Student: Ejub Bilajbegovic. **ID:** 180302019.

Student: Ows Albitar. **ID:** 180302099.

Student: Abdelrahman Enan. **ID:** 170302141.

Fall 2020

Content table

Contents

Abstract

Film-making is a huge industry that attracts the eyes and minds of people all over the world, because it is one of the most entertaining event for people, especially on the weekends and holidays it attracts a huge number of people since it is an entertaining event where people would spend some quality time with their families or friends, and one of the best attractions for huge investors where enormous amount of money flows in the industry in millions. Just looking at the last year the movie "Avengers: Endgame" made around 2.79 billion dollars worldwide. Our goal is to find a model that can predict movies revenue as accurate as possible given information about them from a dataset. Our dataset contains information from IMDb (acronym for Internet movie database) which we downloaded from kaggle [1]. Our goal is to find a numerical value so we used three different models which are suitable for our problem and that are: Linear Regression, Random Forest and Artificial Neural Networks. Working with the models we noticed the differences between them and at the Random Forest Regression had the best result what was 81.25 % after improving it. ANN had the second best accuracy with 77 % and Linear regression had 72 %.

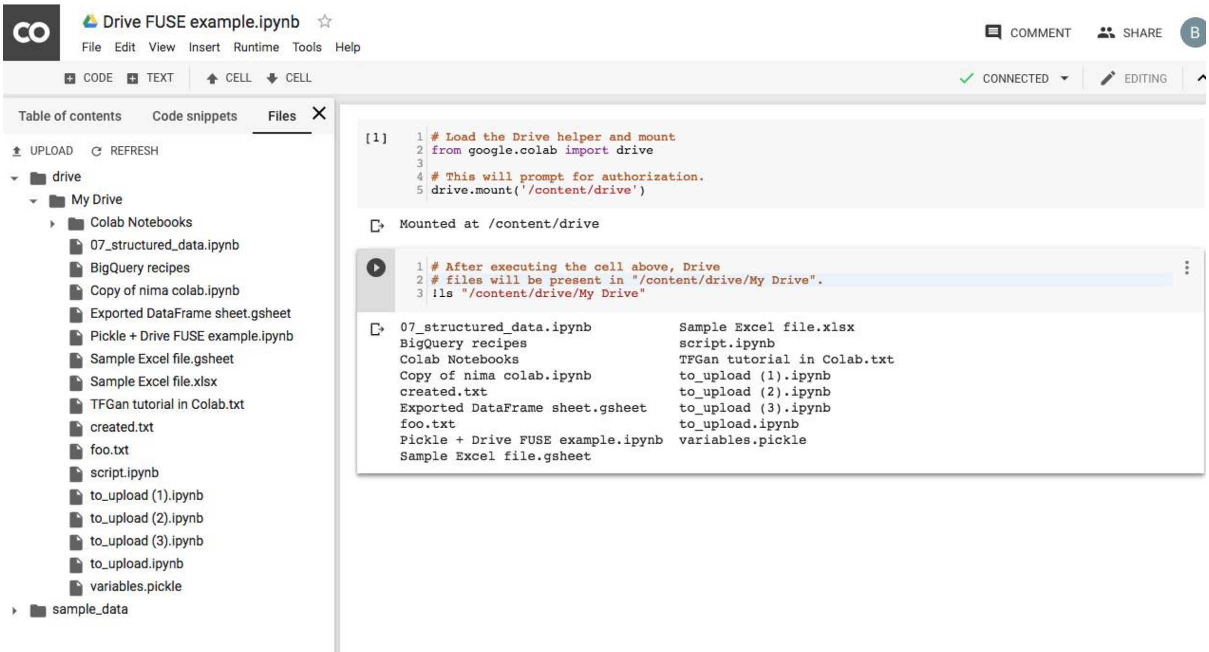
Introduction

Our task is to make a ML model that can predict the worldwide revenue for any movie as accurate as possible given information about it. "Prior to 2020, the global film industry showed healthy projections for the future, with worldwide box office revenue having grown consistently for years and amounting to more than 42 billion U.S. dollars in 2019" [6]. The success of a movie could be stated in terms of revenue, where it depends on various aspects (e.g. genres, actors, budget …), these aspects will be the input data (independent value), based on their values the model should give an estimation of revenue for a movie (dependent value). To predict movie revenue, we explored different datasets which contain a lot and different variables before choosing the current dataset which contains the most variables which gives us a freedom to try different dataset and combinations of the dataset. For the coding part we used python [5] because it is easy to understand and is one of the most common languages in ML also with google Colab a there is great way to work on ML. Since we need to predict a numeric value as our output, our task is clearly a regression task. A detailed information about the steps of the making model process were implemented in details which are shown down below as well as how the dataset was analyzed, prepared, trained, and improved using different types of algorithms to get the aimed accuracy of the model with shown graphs.

Materials and Methods

Programming language, libraries and editor

For our project we decided to use python as our programming language, mostly because it it easy to understand and very readable. Also it is very powerful language and provides a great variety of different libraries and frameworks for machine learning. In our project we used two most known libraries for ML: Scikit learn and keras. For the python programming language, Scikit-learn is a free machine learning software library. It provides several algorithms for grouping, regression, and clustering, including vector support machines, random forest, gradient boosting, k-means … "Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow. It was developed with a focus on enabling fast experimentation" [8]. After deciding what language, we would use we had to choose on which platform/editor we will work. Our main platform was google Colab since we can share the same file and work together on same file. "Colaboratory, or Colab" for short, is a product from Google Research. Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education. More technically, Colab is a hosted Jupyter notebook service that requires no setup to use, while providing free access to computing resources including GPUs" [4]. Our second editor was Jupiterlab which is a web-based editor for machine learning. We used ti through anaconda navigator. Anaconda Navigator, which comes with Anaconda Individual Edition, is a desktop GUI. It makes it simple, without using command-line commands, to open applications and control packages and environments. Below is an example of the google Colab GUI.



Dataset

We used a dataset from the Kaggle website, which is an online platform targeted towards data scientists. The data was collected from the imdb website (an acronym for Internet movie database). IMDB is one of the most popular websites for movies and has millions of readers. The dataset had initially about 45465 row/examples with 27 features/columns, below we can see Looking at the datatypes we have 23 categorical and 4 numerical, but some of them should be changed like the budget which is stored as a string. Below in the left picture we the original dataset before any changes and also all warnings and problems that we had. Some of the warnings can be ignored but other ones are serious and need to be solved. One of the biggest problems was that 83% of values from the revenue were wrong and we had to drop all rows with them. Also we dropped rows with some missing data, but not for every feature, just the important features for which we needed the values.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45466 entries, 0 to 45465
Data columns (total 27 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
 24  cast                   45434 non-null  object
 25  crew                   45434 non-null  object
 26  keywords               45462 non-null  object
dtypes: float64(4), object(23)
memory usage: 9.4+ MB
None
```

Warnings

Dataset has 28 (0.1%) duplicate rows	Warning
belongs_to_collection has 40972 (90.1%) missing values	Missing
belongs_to_collection has a high cardinality: 1699 distinct values	Warning
budget has a high cardinality: 1226 distinct values	Warning
cast has a high cardinality: 42992 distinct values	Warning
crew has a high cardinality: 44635 distinct values	Warning
genres has a high cardinality: 4069 distinct values	Warning
homepage has 37684 (82.9%) missing values	Missing
homepage has a high cardinality: 7674 distinct values	Warning
id has a high cardinality: 45436 distinct values	Warning
imdb_id has a high cardinality: 45418 distinct values	Warning
keywords has a high cardinality: 25990 distinct values	Warning
original_language has a high cardinality: 93 distinct values	Warning
original_title has a high cardinality: 43371 distinct values	Warning
overview has 954 (2.1%) missing values	Missing
overview has a high cardinality: 44308 distinct values	Warning
popularity is an unsupported type, check if it needs cleaning or further analysis	Warning
poster_path has a high cardinality: 45025 distinct values	Warning
production_companies has a high cardinality: 22709 distinct values	Warning
production_countries has a high cardinality: 2394 distinct values	Warning
release_date has a high cardinality: 17337 distinct values	Warning
revenue has 38052 (83.7%) zeros	Zeros
runtime has 1558 (3.4%) zeros	Zeros
spoken_languages has a high cardinality: 1932 distinct values	Warning
tagline has 25054 (55.1%) missing values	Missing
tagline has a high cardinality: 20284 distinct values	Warning
title has a high cardinality: 42276 distinct values	Warning
vote_average has 2998 (6.6%) zeros	Zeros
vote_count has 2899 (6.4%) zeros	Zeros

Them we decided to drop all features that we think and analyzed to be useless. Most of them are either unique (id, imdb_id) or just do not provide any information. Also we have enough data anyway.

```
movie_dataset = movie_dataset.reset_index(drop=True)
movie_dataset.reset_index(drop=True, inplace=True)
movie_dataset= movie_dataset.drop(columns = ['id' , 'imdb_id', 'poster_path','overview','tagline',
                'original_title','title','status','adult','video'])
```

Below we can compare the missing data between the original and new dataset. On the left we have the new dataset, on right the original one. We can see that we only have missing data for two columns in the new one and that is left by intention, because we won’t take the actual value of that features.

```
belongs_to_collection    5903
homepage                 5024
production_countries        0
budget                      0
genres                      0
original_language           0
popularity                  0
production_companies        0
keywords                    0
crew                        0
revenue                     0
runtime                     0
spoken_languages            0
vote_average                0
vote_count                  0
cast                        0
release_date                0
dtype: int64
```

```
belongs_to_collection    40972
homepage                 37684
tagline                  25054
overview                   954
poster_path                386
runtime                    263
release_date                87
status                      87
cast                        32
crew                        32
imdb_id                     17
original_language           11
revenue                      6
spoken_languages             6
title                        6
video                        6
vote_average                 6
vote_count                   6
popularity                   5
keywords                     4
production_countries         3
production_companies         3
id                           0
original_title               0
genres                       0
budget                       0
adult                        0
dtype: int64
```

We divided our process of making our model into four simply steps:

1. Analyzing and exploring the data,
2. Preparing the data for model
3. Training the model
4. Model evaluation and improving the best model

The first three steps we will include in this part of the papers which is “materials and methods”, the last one will be part of the “Results and discussion” where we will deeply speak about how the models evaluated and what are our assumption of how each model works.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7375 entries, 0 to 7374
Data columns (total 17 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   belongs_to_collection  1472 non-null   object 
 1   budget                 7375 non-null   object 
 2   genres                 7375 non-null   object 
 3   homepage               2351 non-null   object 
 4   original_language      7375 non-null   object 
 5   popularity             7375 non-null   object 
 6   production_companies   7375 non-null   object 
 7   production_countries   7375 non-null   object 
 8   release_date           7375 non-null   object 
 9   revenue                7375 non-null   float64
 10  runtime                7375 non-null   float64
 11  spoken_languages       7375 non-null   object 
 12  vote_average           7375 non-null   float64
 13  vote_count             7375 non-null   float64
 14  cast                   7375 non-null   object 
 15  crew                   7375 non-null   object 
 16  keywords               7375 non-null   object 
dtypes: float64(4), object(13)
memory usage: 979.6+ KB
```

Step 1

The first step was importing dataset to workspace and then exploring and analyzing data in general. We found out that our data had a lot of invalid values and also a respectable amount of missing data. For the revenue we excluded the since we need to know the value of our label/dependent value. For some others we decided to let them in and either fill them with the median or to make a ml model which will predict their values and fill it with the predictions. We did that in the case of the budget. The new dataset is above and we can clearly see the difference between this and the original one. After that we removed some attributes which is either unique like (id, imdb_id, poster_path, overview, tagline, original_title, and title) or same of all movies like (states, adult, and video).

Analyzing Numerical data

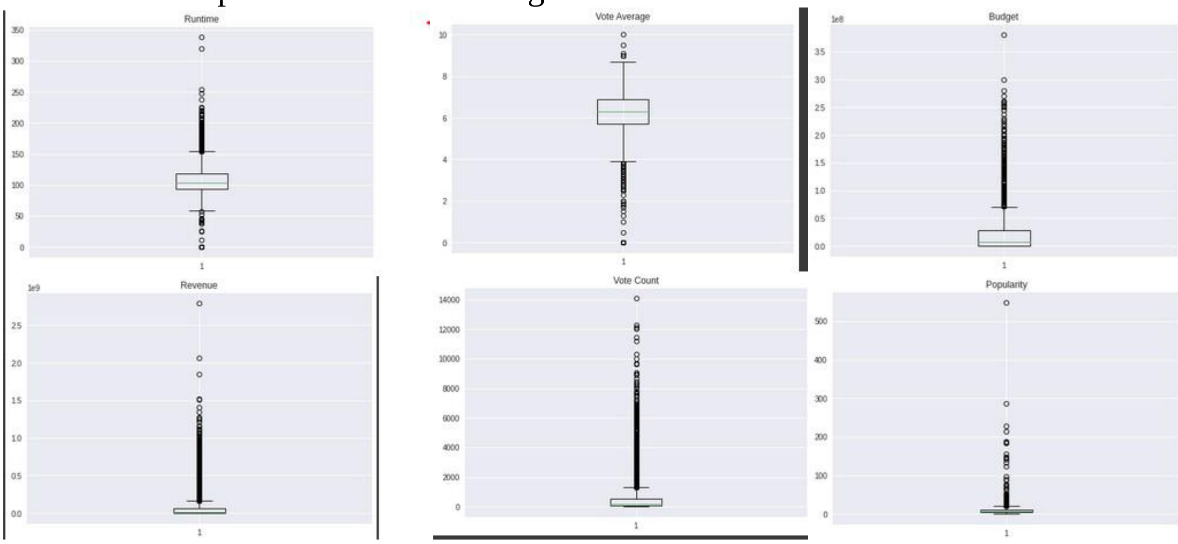
First we started to look at our numerical data. Above we can see that budget and popularity are of object type but need to be numerical, so we first changed that. Then we looked at the summary of all numerical data. Below we can see the summary of vote average and count and revenue. In the case of revenue and vote count the mean is a lot bigger then the median what could be a sign of high number of outliers.

```
count    7375.000000
mean        6.221397
std         1.013761
min         0.000000
25%         5.700000
50%         6.300000
75%         6.900000
max        10.000000
Name: vote_average, dtype: float64
```

```
count     7375.000000
mean       551.517017
std       1087.566705
min          0.000000
25%         41.000000
50%        159.000000
75%        538.500000
max      14075.000000
Name: vote_count, dtype: float64
```

```
count    7.375000e+03
mean     6.779991e+07
std      1.440350e+08
min      1.000000e+00
25%      2.395116e+06
50%      1.670286e+07
75%      6.657980e+07
max      2.787965e+09
Name: revenue, dtype: float64
```

Next we see the summary of budget, runtime and popularity. All numerical data looks good balanced, we just need to take care of the zeros like in the budget and runtime because it is impossible to have 0 budget and 0 runtime.

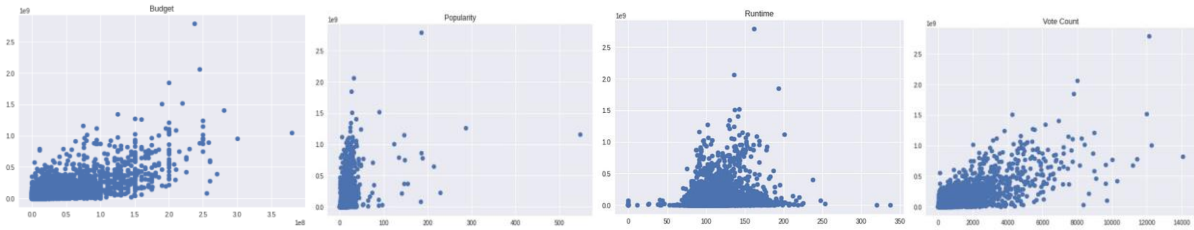


Below we can clearly see how many outliners each numerical feature has. And also what is the correlation between each numerical feature to the dependent/label value.

```
budget          614
popularity      194
revenue         837
runtime         274
vote_average    132
vote_count      858
dtype: int64
```

- Correlations :
  1. Vote count : 77.5 %
  2. Budget : 74.2 %
  3. Popularity : 45.8 %
  4. Runtime : 19.8 %
  5. Vote average : 14.6 %

Looking at the scatterplot we can see that vote count and budget are correlated to revenue. Especially the runtime doesn't look correlated to the revenue.



Analyzing categorical data

The analyzing of the categorical data was impossible before the data preparation cause most data was stored in string and some were even json formats stored in a string. So this could also be a part of data preparation but because this I important to know at this stage of the process we decided to put it in the data analysis part. Below is the code to change from text to a dictionary [10], so we can easily extract the wanted variable from our dictionary.

```
# For these features we change datatype to a dictionary instead of a string type
text_cols = ['belongs_to_collection', 'genres', 'production_companies',
             'production_countries', 'spoken_languages', 'keywords', 'cast', 'crew']

def text_to_dict(df):
    for col in text_cols:
        df[col] = df[col].apply(lambda x: {} if pd.isna(x) else ast.literal_eval(x))
    return df

movie_dataset = text_to_dict(movie_dataset)
```

```
# A example of the belongs to colection feature
movie_dataset['belongs_to_collection'][0]
```

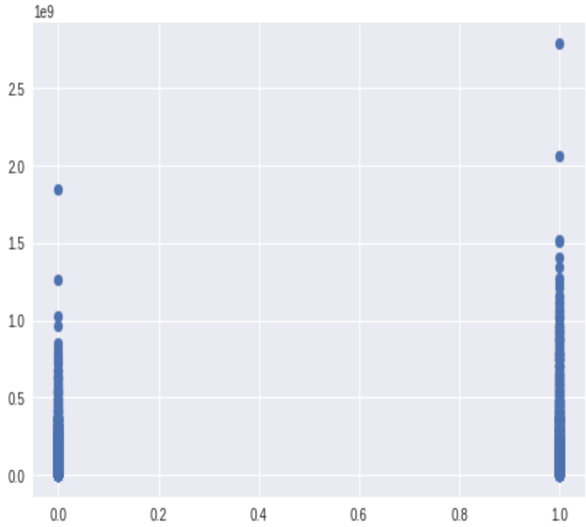
```
{'backdrop_path': '/9FBwqcd9IRruEDUrTdcaafOMKUq.jpg',
 'id': 10194,
 'name': 'Toy Story Collection',
 'poster_path': '/7G9915LfUQ2lVfwMEEhDsn3kT4B.jpg'}
```

This above is an example output of a dictionary, by calling id we would get the id value. After every categorical data is in that datatype we can start to extract the information that we want. To better understand the process of data cleaning we need to explain the differences between our categorical data we:

1. We have simple categorical data which store just one value in one row, like original language, belongs to collected, homepage
2. We have categorical data which stores multiple classes in one row, like spoken language, cast, crew … For those data we will either extract all names or just important ones
3. We have release date which is date type

Belongs to collection and homepage

First we handled the belongs to collection and homepage feature, because we thought that the values are very unique and encoding it won't help our model we decided to make it a binary like value, which will store a 1 in case the movie contains a homepage/collection or 0 if not. For the collection case the new value had a correlation above 30 % and for the homepage above 24 % what we think is better than encoding that values of that 2 features. On the left we have a scatter plot between homepage and the revenue. We used the binary version from now.



Genres, Cast, Production companies and Production countries

All of those columns share the same initial type of value all are list which store different dictionaries. All of them except the cast we will extract all names from the dictionaries just for the cast we will extract just the top 3 actor names. Extracting all cast names would be too much columns to encode. This below is an example of a code which extracts the keywords. Down we will have three different list one with the number of keywords, names of keywords and keywords stored in a vector. We store it in a vector so we can get the number of classes for each categorical data

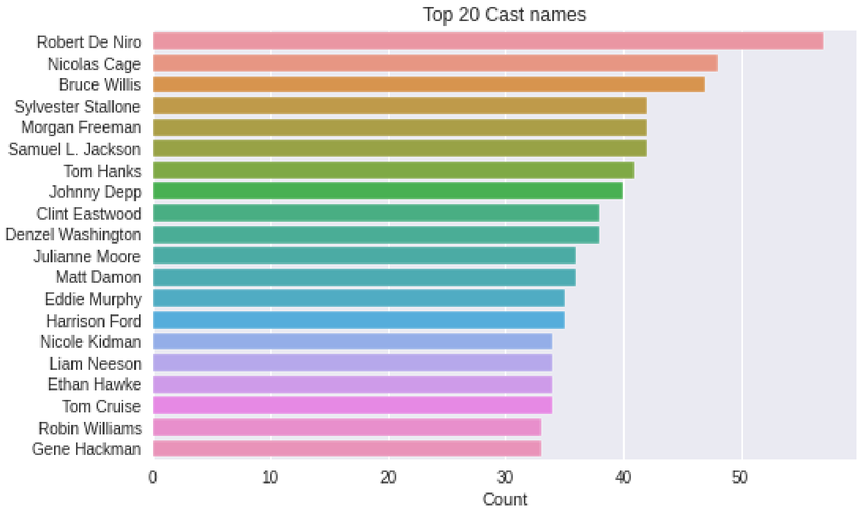
```
allgenres =[]
def getgenres(row):
    x = []
    for i in row:
        allgenres.append(i['name'])
        x.append(i['name'])
    return x

genr = movie_dataset['genres'].apply(lambda x: getgenres(x))
gen_num = movie_dataset.genres.apply(lambda x: len(x))
```

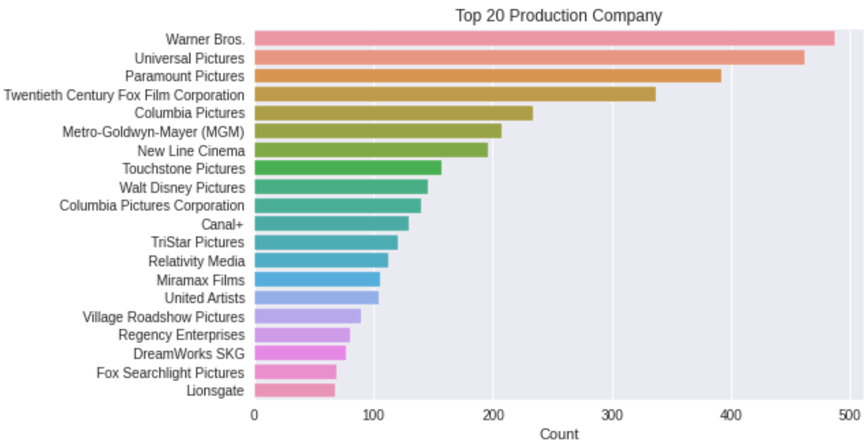
Just for the cast this code is different cause we had to specify the order of the actor below we can see the code

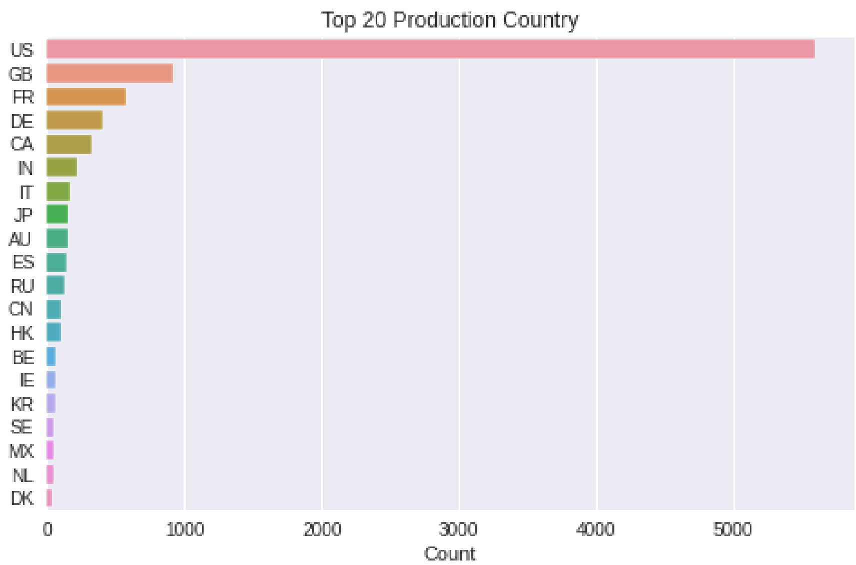
```
def getactorsformovie(row):
    x = []
    for i in row:
        if i['order'] in (0,1,2):
            x.append(i['name'])
    return x

castpeople =[]

for row in movie_dataset['cast']:
    for single in row:
        if single['order'] in (0,1,2):
            castpeople.append(single['name'])

castnames = movie_dataset['cast'].apply(lambda x: getactorsformovie(x))
```

When we finished the process outputs looks like the picture down:

```
] castnames.head(5)

0              [Tom Hanks, Tim Allen, Don Rickles]
1       [Robin Williams, Jonathan Hyde, Kirsten Dunst]
2    [Whitney Houston, Angela Bassett, Loretta Devine]
3           [Steve Martin, Diane Keaton, Martin Short]
4              [Al Pacino, Robert De Niro, Val Kilmer]
Name: cast, dtype: object
```

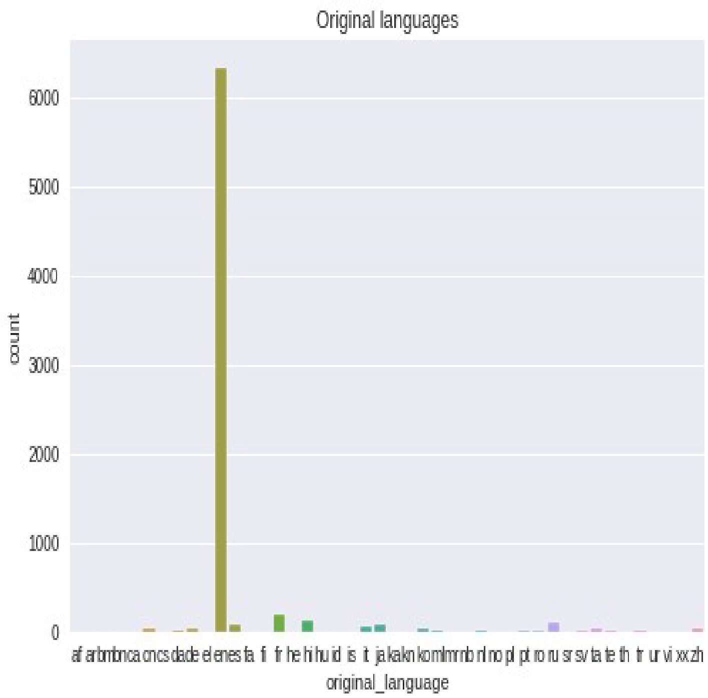






Crew and original language

Although crew is containing lists of dictionaries for every crew member we decided to just extract the director's names so we will have a vector of names at the end and original language and the cleaned crew will be of same type. To clean the crew variable, we just had to iterate throughout the list until we found director value for job key and then to extract the name value. For the original language we didn't have to do any cleaning. Below we can see the histogram of original language. As noticed most movies are filmed on English. "en" is the mode and its count is 6329 values.



Feature name	Label number
Cast	8328
Crew	3358
Prod Countries	98
Prod Companies	7073
Original language	44
Genres	20

Adding and subtracting features

Analyzing the cast and crew we noticed that the crew-number and cast-number have a correlation of 40.3 % and 37.3 %. So we decided to include it in our new dataset. Also we realized that keywords and spoken languages are useless and we won't need them in our dataset so we excluded them.

Step 2

The second step was the data preparation, and here we filled the invalid numerical number with the median except for the budget, where we trained a random forest model to predict the missing values for budget so we can use it. We used a simple random forest regression model we didn't want it to be to complex, below we have the code for it:

```
# We will predict values for budget where budget value is 0

train_bud = copy.copy(movie_dataset[movie_dataset.budget != 0])
unknown_bud = copy.copy(movie_dataset[movie_dataset.budget == 0])
Y = train_bud.budget

X = train_bud[[ 'vote_count'  ,'popularity' , 'homepage' , 'revenue' ,
               'crew_number' , 'cast_number'  ,'release_year']]

newX = unknown_bud[['vote_count'  ,'popularity' , 'homepage' , 'revenue' ,
                    'crew_number' , 'cast_number'  ,'release_year' ]]



rfr = RandomForestRegressor(n_estimators= 100)
rfr.fit(X, Y)

predicted = rfr.predict(newX)
movie_dataset.budget[movie_dataset.budget == 0] = predicted
```

The next things are to encode our categorical data and to scale the numerical features. The biggest problem was that we had a lot of column names that were repeating so we had to either manually rename them or written a code for that. For the crew feature we wrote a code which will add "_dir" at the end of every name which is in the cast columns names list so that we won't have duplicates. For some names we manually renamed them. After the encoding the dataset consists of 18 950 columns what is a large number. To understand how a number like that effects our models we will make a small dataset with just the most important features. We will just choose features with a greater correlation the 15 % or a lower than – 15%. Because we will use more than one model we decided to use scaled and unscaled dataset and to try the difference between normalization and standardization. Now we have two datasets a large one and a small one. Below we have the info output of our large dataset.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7375 entries, 0 to 7374
Columns: 18950 entries, belongs_to_collection to 12
dtypes: Sparse[int64, 0](15519), Sparse[uint8, 0](3420), float64(5), int64(6)
memory usage: 1.6 MB
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7375 entries, 0 to 7374
Data columns (total 28 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   belongs_to_collection     7375 non-null   int64
 1   budget                    7375 non-null   int64
 2   homepage                  7375 non-null   int64
 3   popularity                7375 non-null   float64
 4   revenue                   7375 non-null   float64
 5   runtime                   7375 non-null   float64
 6   vote_count                7375 non-null   float64
 7   crew_number               7375 non-null   int64
 8   cast_number               7375 non-null   int64
 9   Action                    7375 non-null   Sparse[int64, 0]
 10  Adventure                 7375 non-null   Sparse[int64, 0]
 11  Animation                 7375 non-null   Sparse[int64, 0]
 12  Drama                     7375 non-null   Sparse[int64, 0]
 13  Family                    7375 non-null   Sparse[int64, 0]
 14  Fantasy                   7375 non-null   Sparse[int64, 0]
 15  US                        7375 non-null   Sparse[int64, 0]
 16  DreamWorks Animation      7375 non-null   Sparse[int64, 0]
 17  Heyday Films              7375 non-null   Sparse[int64, 0]
 18  Lightstorm Entertainment  7375 non-null   Sparse[int64, 0]
 19  Marvel Studios            7375 non-null   Sparse[int64, 0]
 20  Pixar Animation Studios   7375 non-null   Sparse[int64, 0]
 21  Walt Disney Pictures      7375 non-null   Sparse[int64, 0]
 22  Daisy Ridley              7375 non-null   Sparse[int64, 0]
 23  Emma Watson               7375 non-null   Sparse[int64, 0]
 24  Ian McKellen              7375 non-null   Sparse[int64, 0]
 25  Rupert Grint              7375 non-null   Sparse[int64, 0]
 26  David Yates               7375 non-null   Sparse[uint8, 0]
 27  James Cameron_dir         7375 non-null   Sparse[uint8, 0]
dtypes: Sparse[int64, 0](17), Sparse[uint8, 0](2), float64(4), int64(5)
memory usage: 683.5 KB
```

Step 3

Our thirst step was the training of all ml models using two different datasets. In this step we will just train them without writing about the results. For the validation we used the holdout method and for the accuracy the r squared score.

Linear regression

A special case of regression analysis, a mathematical method that seeks to describe an observable dependent variable in terms of one or more independent variables. It tries to fit the best parameters for a function which describes best the output values. In our case we use the multiple linear regression because we have more than one independent variable. Below is our code:

Large dataset

```
linear_regressor = LinearRegression()
linear_regressor.fit(X_train_large, Y_train_large)
accuracy = linear_regressor.score(X_test_large, Y_test_large)
print('Accuracy: %.2f' % (accuracy*100))
```

Small dataset

```
[174] linear_regressor.fit(X_train_small, Y_train_small)
      accuracy = linear_regressor.score(X_test_small, Y_test_small)
      print('Accuracy: %.2f' % (accuracy*100))
```

Random Forest Regression

Random Forest Regression is a supervised learning algorithm that uses ensemble learning method for regression. Ensemble learning method is a technique that combines predictions from multiple machine learning algorithms to make a more accurate prediction than a single model. When training the data, we have multiple different parameters. The n_estimators is the number of trees in the forest. Usually with a higher number our accuracy is better but increasing the number means that we have more to wait to train the model. Random_state is for the randomness of the model. Below is our code:

Large dataset

```
regressor = RandomForestRegressor(n_estimators= 1000 , random_state= 0 )
regressor.fit(X_train_large,Y_train_large)
predicted = regressor.predict(X_test_large)
r2_score(Y_test_large , predicted)
```

Small dataset

```
regressor = RandomForestRegressor(n_estimators= 100 , random_state= 0 )
regressor.fit(X_train_small,Y_train_small)
predicted = regressor.predict(X_test_small)
r2_score(Y_test_small , predicted)
```

Artificial Neuronal Network

An artificial neural network (ANN) is a computer system developed to replicate how information is analyzed and interpreted by the human brain. It is the basis of artificial intelligence (AI) and solves problems which, by human or mathematical standards, would prove impossible or challenging. For the implementation of the code we used the keras library which is widely known between python data scientists. With the keras library we have manually define our model. We have to specify to number of layers, number of nodes in one layer, type of layers, type of output layer etc. After making a sequential object we can add layer to it, first we add our input layer which also contains the characteristic of the first hidden layer, we have to specify the input feature number. The number besides that number are the modes number and activation = "Relu" is the activation function. "Relu" stands for "rectified linear unit activation function". Then we continued to add our hidden layers, for which also have the same activation function. Our last layer contains ta different activation function in fact "linear" which is used for numerical output. Before training it we need to compile it as the loss argument we used mean squared error, next the optimizer = "**adam**" means that "We define the **optimizer** as the efficient stochastic gradient descent algorithm "**adam**". This is a popular version of gradient descent because it automatically tunes itself and gives good results in a wide range of problems." [14].

Then we van train the model, verbose = 0 means that we will not see the progress bars in the output and epochs stands for "One pass through all of the rows in the training dataset." [14]. Below we can see our code for ANN models, the only difference between the two models is that we have more nodes in the hidden layers and more input features.

```python
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model.add(Dense(40, input_dim=27,
                activation='relu'))
model.add(Dense(30, activation='relu'))
model.add(Dense(30, activation='relu'))
model.add(Dense(30, activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(1, activation='linear'))
model.compile(loss='mse', optimizer='adam')
model.fit(X_train_std, Y_train_small,
          epochs=500, verbose=0)
predicted = model.predict(X_test_std)
r2_score(Y_test_small, predicted)
```

```python
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model.add(Dense(100, input_dim=18949,
                activation='relu'))
model.add(Dense(200, activation='relu'))
model.add(Dense(500, activation='relu'))
model.add(Dense(200, activation='relu'))
model.add(Dense(100, activation='relu'))
model.add(Dense(1, activation='linear'))
model.compile(loss='mse', optimizer='adam')
model.fit(X_train_std, Y_train_large ,
          epochs=1000, verbose=0)
predicted = model.predict(X_test_std)
r2_score(Y_test_large , predicted)
```

Results and Discussion

Step 4

Our fourth Step is similar to results and discussion, because we evaluated and then improved our best model in this step. Next we will show in a table all results that we gathered throughout this process. As mentioned above we used the holdout validation and r squared score for all models.

Linear Regression

	Small dataset	Large dataset
Unscaled dataset	71.5 %	Wrong model
Normalized	71.5 %	Wrong model
Standardized	69.6 %	Wrong model

The best accuracy for the linear regression was 71.5 %, the only difference was that it needed less time for the normalized data but anything else is the same. The standardized data was a little bit worse than the other two. The reason why I put wrong model is that the output was -5556, but it should be between 0 – 100 so what we later found out that it is a sign that we did something terribly wrong. Our assumption is that too much column and especially useless effects negatively on the model and that we need to take care just to use the most important features. But also the best case for linear regression is worse than the other two models so it is good and fast but it has problems with complex data.

Random Forest Regression

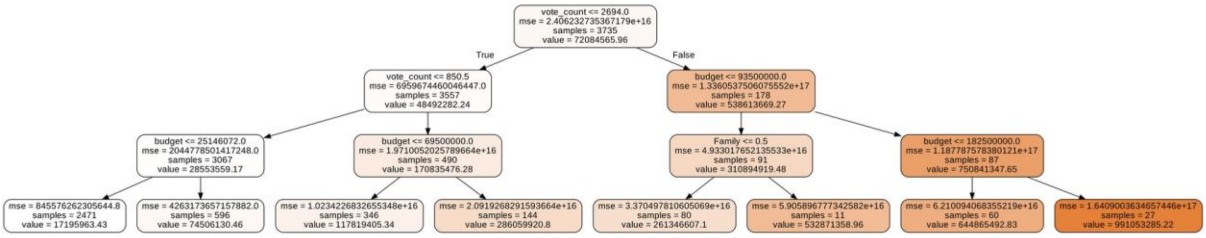
	Small dataset	Large dataset
Unscaled dataset	77.4 %	80.1 %
Normalized	77.3 %	80.1 %
Standardized	74.9 %	78.5 %

The best accuracy for the random forest was 80.1 % and that for the large dataset with over 18 thousand columns. Here we can clearly see that random forest is the best model out of my 3 models for large datasets. The training time is between the Ann and Linear regression, with Ann being the slowest one to train. Similar to the previous table the standardization was little bit worse in accuracy than unscaled and normalization. Here the model trained with large dataset did better by less than 3 %. Below we can see an example of a smaller random forest regression tree. The trees are hard to analyze and understand because we have more than 20 levels which make a huge tree and impossible to understand easily. Because this has the best accuracy of our 3 models we will after the ANN evaluation improve this model

ANN

	Small dataset	Large dataset
Normalized	77.2 %	43.5 %

Standardized	75.6 %	47.1 %




```
### Small node number ###

Number : 1 r2score : 0.6789845733203297
Number : 2 r2score : 0.7016334424479822
Number : 3 r2score : 0.7047282210879242
Number : 4 r2score : 0.7141346143636231
Number : 5 r2score : 0.7563318772627833
Number : 6 r2score : 0.746477796175834
Number : 7 r2score : 0.7140102002829531
Number : 8 r2score : 0.7255701927222327
Number : 9 r2score : 0.7066623924297952

### Large node number ###

Number : 1 r2score : 0.6978276178049678
Number : 2 r2score : 0.7028582374695085
Number : 3 r2score : 0.7184415190900537
Number : 4 r2score : 0.716230999452663
Number : 5 r2score : 0.7133154414830811
Number : 6 r2score : 0.6532854637063041
Number : 7 r2score : 0.6731633598260169
Number : 8 r2score : 0.6079540411107791
Number : 9 r2score : 0.6116076016162988
```

Because we cannot train the model with the unscaled data we decided to exclude that row in this table. Similar to the linear regression the model did better with the small dataset than the large one by around 30 %. The ANN model is the second best by accuracy. Also in all models the normalized accuracy did better. The time needed to train a model was the longest one. Especially in ANN where we need to adjust a lot of parameters it is quite exhausting waiting until it finishes. Running multiple different models with different number of layers and different node number we decided to use 5 hidden layers. Below we can see the output of a function which generated different ANN models and their accuracy. The first block are models with a low node number from 1 to 9 hidden layers, the second block are models with a high number of node numbers from 1 to 9 hidden layers. So we decided to use small number of nodes and 5 hidden layers.

Improving Random Forest Regression with hyper-parameters

1. With the next four steps we can improve our model accuracy:
2. Specify the maximum depth of the trees
3. Increase or decrease the number of estimators
4. Specify the maximum number of features to be included at each node split

Increase or decrease the number for minimum of samples required to be at a leaf node Similar as the ANN model we trained different model to best understand how each parameter effects our accuracy. Below we can see the results

```
Less number of trees in the forest : 77.29

More number of trees in the forest : 77.69

Smaller number for minimum of samples required to be at a leaf node : 77.59

Larger number for minimum of samples required to be at a leaf node : 77.59

Number of features to consider when looking for the best split is 0.5 : 79.26

Number of features to consider when looking for the best split is 1 : 76.84

Number of features to consider when looking for the best split is sqrt : 79.58

Number of features to consider when looking for the best split is log2 : 79.37

Maximal depth of tree is 5 : 76.80

Maximal depth of tree is 20 : 77.78

Maximal depth of tree is 100 : 77.66
```

The previous accuracy for the small dataset was 77.2 % by changing the number of trees and features splitting we can increase the accuracy. The maximal depth is not bad but doesn't improves the model by that much, after applying that on both models we improved the small dataset accuracy by 2.5 % and the large dataset model by 1 % what we can see below.

Small dataset →

```
regressor = RandomForestRegressor(n_estimators= 1000 ,
                                  max_features=0.5)

regressor.fit(X_train_large,Y_train_large)
predicted = regressor.predict(X_test_large)
print('The improved model for RFR : %.2f'
      % (100*r2_score(Y_test_large , predicted)))
```

```
The improved model for RFR : 81.25
```

Large dataset →

```
regressor = RandomForestRegressor(n_estimators= 1000 ,
                                  min_samples_leaf=1 ,
                                  max_features='sqrt' ,
                                  max_depth= 20 )

regressor.fit(X_train_norm,Y_train_small)
predicted = regressor.predict(X_test_norm)
print('The improved model for RFR : %.2f'
      % (100*r2_score(Y_test_small , predicted)))
```

```
The improved model for RFR : 79.70
```

Conclusion

In a nutshell, the model was successfully implemented using python programming language. We analyzed, cleaned and prepared our dataset for our three machine learning models (Linear regression, Random forest regression and ANN). We can say with confidence that our RFR model has an accuracy above 80 % and that we learned a lot about python, data, ML models. Especially the problem solving was great in our team. A major finding which was obtained in this specific model was that Random Forest Regression had the best accuracy out of all algorithms used with an accuracy of 81.25 % which allowed us to obtain an overall great results of the model. Going into this project we all thought that ANN will have the best accuracy but we found out that all models are really data dependent and that we cannot be 100% sure how some model will perform. Our team worked 80% of the time together and we managed to meet twice a week in person despite the current situation. We solved all tasks together and brainstormed all possible solutions. A special thanks goes to Assist.Prof.Dr. Kanita Karadjuzovic-Hadziabdic for consultation and putting efforts throughout the process of the project.

Team member name	**Specific tasks completed**
Ejub Bilajbegovic	Model training and evaluation , presentation
Abdelrahman Enan	Data preparation and Model training, presentation
Ows Albitar	Analyzing data and most of project paper

References

1. https://www.kaggle.com/rounakbanik/the-movies-dataset
2. https://www.boxofficemojo.com/
3. https://scikit-learn.org/stable/
4. https://research.google.com/colaboratory/faq.html
5. https://becominghuman.ai/best-languages-for-machine-learning-in-2020-6034732dd242
6. https://www.statista.com/topics/964/film/
7. https://www.geeksforgeeks.org/top-5-best-programming-languages-for-artificialintelligence- field/
8. https://keras.io/about/
9. https://www.anaconda.com/products/individual
10. https://www.w3schools.com/python/python_dictionaries.asp
11. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html
12. https://medium.com/swlh/random-forest-and-its-implementation-71824ced454f
13. https://www.forbes.com/sites/bernardmarr/2018/09/24/what-are-artificial-neuralnetworks- a-simple-explanation-for-absolutely-anyone/?sh=41d36ca11245
14. https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/