**Predicting IMDb Scores Using Machine Learning**

**TEAM MEMBER**

**732521104048 : SUBHASHINI P**

**Phase 5 Submission Document**

**Project : Predicting IMDb Scores**

**Problem Definition:**

The problem is to develop a machine learning model that predicts IMDb scores of movies available on Films based on features like genre, premiere date, runtime, and language. The objective is to create a model that accurately estimates the popularity of movies, helping users discover highly rated films that match their preferences. This project involves data preprocessing, feature engineering, model selection, training, and evaluation .

**Design Thinking:**

1. **Data Source:** Utilize a dataset containing information about movies, including features like genre, premiere date, runtime, language, and IMDb scores.
2. **Data Preprocessing:** Clean and preprocess the data, handle missing values, and convert categorical features into numerical representations.
3. **Feature Engineering:** Extract relevant features from the available data that could contribute to predicting IMDb scores.
4. **Model Selection:** Choose appropriate regression algorithms (e.g., Linear Regression, Random Forest Regressor) for predicting IMDb scores.
5. **Model Training:** Train the selected model using the preprocessed data.
6. **Evaluation:** Evaluate the model's performance using regression metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared.

**Working Methodology:**

The working method for this work involves few steps. The methodology is shown in figure 1. The steps are described below.

• Data Extraction

• Data Preprocessing

• Applying Machine Learning Techniques

• Comparing the results of different algorithms

DATA EXTRACTION

DATA PREPROCESSING

MACHINE LEARNING TECHNIQUES

**Algorithm :**

Algorithm for developing the model

1: Prepare data set

2: Check Minority

3: If needed apply SMOTE algorithm until the minority class becomes equal to the size of it’s closest class 4: Classification

5: Accuracy ←− 0

6: while True do

7: Resample Data

8: Call (Classifier)

9: if % of correctly classified Instance >Previous Accuracy Measure then

10: Accuracy ←− % of correctly classif ied Instance

11: else

12: Break

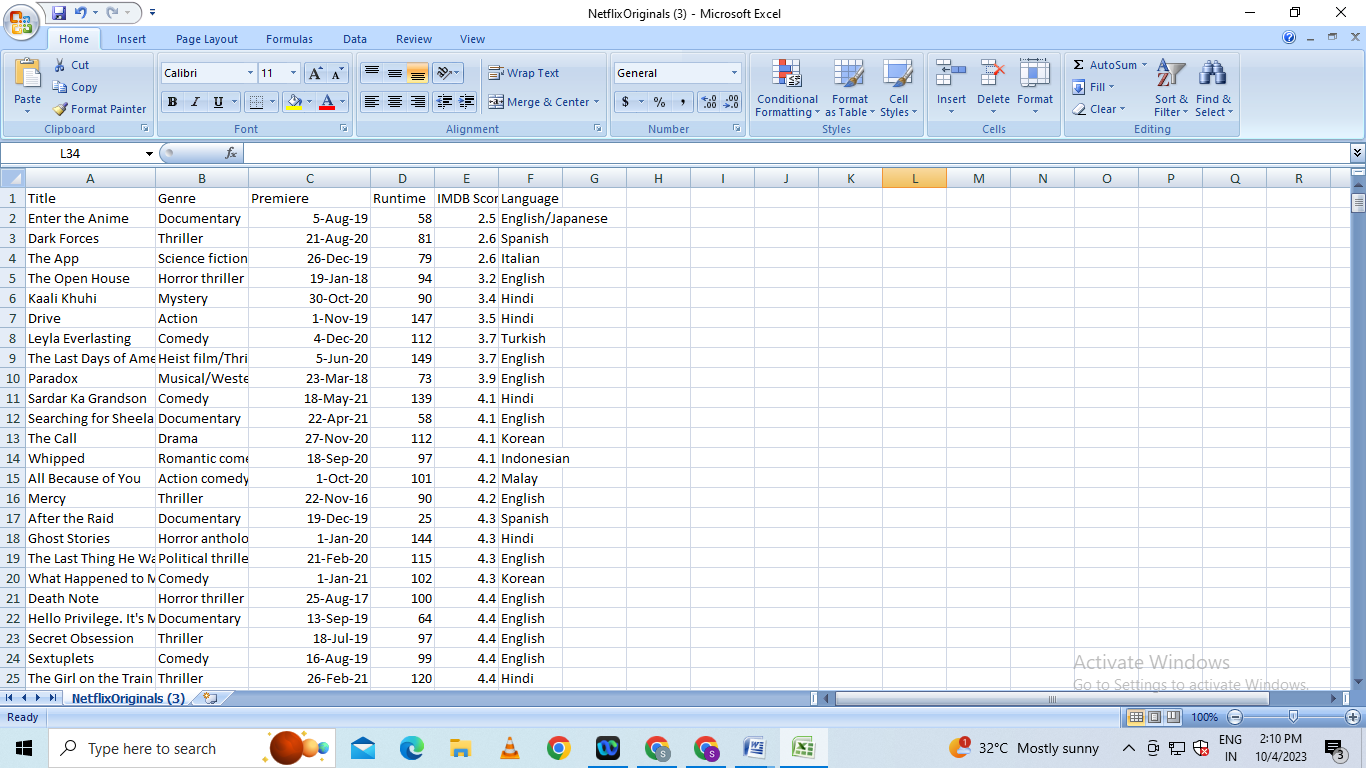
13: end if

14: end while=0

**Data Source:**

A Good Data for Predicting IMDb Scores using machine learning model should be Accurate , complete , covering the geographic area of interest , accessible

Dataset Link : **(**[**https://www.kaggle.com/datasets/luiscorter/netflix-original-films-imdb-scores**](https://www.kaggle.com/datasets/luiscorter/netflix-original-films-imdb-scores))



**Data Preprocessing:**

Data preprocessing is the critical first step in any machine learning project.It involves cleaning the data,removing outliers and handling missing values to prepare the dataset for model training. In the context of the predicting the IMDB scores project , let’s elaborate on the specific steps:

1. **Duplicate Removal:**

Duplicate rows can introduce bias into model.We will identify and remove duplicates,typically by sorting the dataset based on unique identifier and then eliminating consecutive rows with same identifiers

**b)Handling Missing Values:**

Missing data is common and needs to be addressed . We will utilize suitable methods such as :

* **Mean Imputation**
* **Median Imputation**

## Imports

*# This Python 3 environment comes with many helpful analytics libraries installed*

*# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python*

*# For example, here's several helpful packages to load*

import numpy as np *# linear algebra*

import pandas as pd *# data processing, CSV file I/O (e.g. pd.read\_csv)*

*# Input data files are available in the read-only "../input/" directory*

*# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory*

import os

for dirname, \_, filenames **in** os.walk('/kaggle/input'):

for filename **in** filenames:

print(os.path.join(dirname, filename))

*# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"*

*# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session*

/kaggle/input/netflix-original-films-imdb-scores/NetflixOriginals.csv

In [2]:

import matplotlib.pyplot as plt

import seaborn as sns

import plotly.express as px

from datetime import datetime,timedelta

**Dataset**

In [3]:

ds = pd.read\_csv("/kaggle/input/netflix-original-films-imdb-scores/NetflixOriginals.csv",encoding = "ISO-8859-1")

ds\_date = ds.copy()

ds.head(5)

Out[3]:

|  | Title | Genre | Premiere | Runtime | IMDB Score | Language |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | Enter the Anime | Documentary | August 5, 2019 | 58 | 2.5 | English/Japanese |
| 1 | Dark Forces | Thriller | August 21, 2020 | 81 | 2.6 | Spanish |
| 2 | The App | Science fiction/Drama | December 26, 2019 | 79 | 2.6 | Italian |
| 3 | The Open House | Horror thriller | January 19, 2018 | 94 | 3.2 | English |
| 4 | Kaali Khuhi | Mystery | October 30, 2020 | 90 | 3.4 | Hindi |

In [4]:

ds.describe().T

Out[4]:

|  | count | mean | std | min | 25% | 50% | 75% | max |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Runtime | 584.0 | 93.577055 | 27.761683 | 4.0 | 86.0 | 97.00 | 108.0 | 209.0 |
| IMDB Score | 584.0 | 6.271747 | 0.979256 | 2.5 | 5.7 | 6.35 | 7.0 | 9.0 |

insights: categorical of IMDB Score 5.7 > rendah 6.35 > sedang 7.0 > tinggi 9.0 > sangat tinggi

In [5]:

ds.info(verbose=True,show\_counts=True)

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 584 entries, 0 to 583

Data columns (total 6 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Title 584 non-null object

1 Genre 584 non-null object

2 Premiere 584 non-null object

3 Runtime 584 non-null int64

4 IMDB Score 584 non-null float64

5 Language 584 non-null object

dtypes: float64(1), int64(1), object(4)

memory usage: 27.5+ KB

In [6]:

ds.isna().sum()

Out[6]:

Title 0

Genre 0

Premiere 0

Runtime 0

IMDB Score 0

Language 0

dtype: int64

In [7]:

ds['Title'].value\_counts()

Out[7]:

Enter the Anime 1

Have a Good Trip: Adventures in Psychedelics 1

Tallulah 1

The Old Guard 1

Tony Robbins: I Am Not Your Guru 1

..

Cam 1

Earthquake Bird 1

Frankenstein's Monster's Monster, Frankenstein 1

Horse Girl 1

David Attenborough: A Life on Our Planet 1

Name: Title, Length: 584, dtype: int64

In [8]:

ds['Genre'].value\_counts()

Out[8]:

Documentary 159

Drama 77

Comedy 49

Romantic comedy 39

Thriller 33

...

Romantic comedy-drama 1

Heist film/Thriller 1

Musical/Western/Fantasy 1

Horror anthology 1

Animation/Christmas/Comedy/Adventure 1

Name: Genre, Length: 115, dtype: int64

In [9]:

ds['Premiere'].value\_counts()

Out[9]:

October 2, 2020 6

November 1, 2019 5

October 18, 2019 5

November 2, 2018 4

June 19, 2020 4

..

September 20, 2019 1

March 10, 2017 1

March 17, 2017 1

May 29, 2015 1

October 4, 2020 1

Name: Premiere, Length: 390, dtype: int64

In [10]:

ds\_date["Premiere"] = ds\_date["Premiere"].apply(lambda x: "".join(x for x **in** x.replace(".",",")))

ds\_date["PremiereDate"] = ds\_date["Premiere"].apply(lambda x: datetime.strptime(x, "%B **%d**, %Y").date())

ds\_date["Year"] = ds\_date["Premiere"].apply(lambda x: "".join(x for x **in** x.replace(",","").split()[-1]))

*#Convert object to date*

ds\_date["PremiereDate"] = pd.to\_datetime(ds\_date["PremiereDate"])

ds\_date

Out[10]:

|  | Title | Genre | Premiere | Run time | IMDB Score | Language | PremiereDate | Year |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | Enter the Anime | Documentary | August 5, 2019 | 58 | 2.5 | English/Japanese | 2019-08-05 | 2019 |
| 1 | Dark Forces | Thriller | August 21, 2020 | 81 | 2.6 | Spanish | 2020-08-21 | 2020 |
| 2 | The App | Science fiction/Drama | December 26, 2019 | 79 | 2.6 | Italian | 2019-12-26 | 2019 |
| 3 | The Open House | Horror thriller | January 19, 2018 | 94 | 3.2 | English | 2018-01-19 | 2018 |
| 4 | Kaali Khuhi | Mystery | October 30, 2020 | 90 | 3.4 | Hindi | 2020-10-30 | 2020 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 579 | Taylor Swift: Reputation Stadium Tour | Concert Film | December 31, 2018 | 125 | 8.4 | English | 2018-12-31 | 2018 |
| 580 | Winter on Fire: Ukraine's Fight for Freedom | Documentary | October 9, 2015 | 91 | 8.4 | English/Ukranian/Russian | 2015-10-09 | 2015 |
| 581 | Springsteen on Broadway | One-man show | December 16, 2018 | 153 | 8.5 | English | 2018-12-16 | 2018 |
| 582 | Emicida: AmarElo - It's All For Yesterday | Documentary | December 8, 2020 | 89 | 8.6 | Portuguese | 2020-12-08 | 2020 |
| 583 | David Attenborough: A Life on Our Planet | Documentary | October 4, 2020 | 83 | 9.0 | English | 2020-10-04 | 2020 |

584 rows × 8 columns

In [11]:

ds\_date.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 584 entries, 0 to 583

Data columns (total 8 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Title 584 non-null object

1 Genre 584 non-null object

2 Premiere 584 non-null object

3 Runtime 584 non-null int64

4 IMDB Score 584 non-null float64

5 Language 584 non-null object

6 PremiereDate 584 non-null datetime64[ns]

7 Year 584 non-null object

dtypes: datetime64[ns](1), float64(1), int64(1), object(5)

memory usage: 36.6+ KB

In [12]:

ds['Language'].value\_counts()

Out[12]:

English 401

Hindi 33

Spanish 31

French 20

Italian 14

Portuguese 12

Indonesian 9

Japanese 6

Korean 6

German 5

Turkish 5

English/Spanish 5

Polish 3

Dutch 3

Marathi 3

English/Hindi 2

Thai 2

English/Mandarin 2

English/Japanese 2

Filipino 2

English/Russian 1

Bengali 1

English/Arabic 1

English/Korean 1

Spanish/English 1

Tamil 1

English/Akan 1

Khmer/English/French 1

Swedish 1

Georgian 1

Thia/English 1

English/Taiwanese/Mandarin 1

English/Swedish 1

Spanish/Catalan 1

Spanish/Basque 1

Norwegian 1

Malay 1

English/Ukranian/Russian 1

Name: Language, dtype: int64

EDA

In [13]:

ds['Genre'].value\_counts()

genre = ds['Genre'].value\_counts()

genre.head()

Out[13]:

Documentary 159

Drama 77

Comedy 49

Romantic comedy 39

Thriller 33

Name: Genre, dtype: int64

In [14]:

plt.figure(figsize=(16, 5))

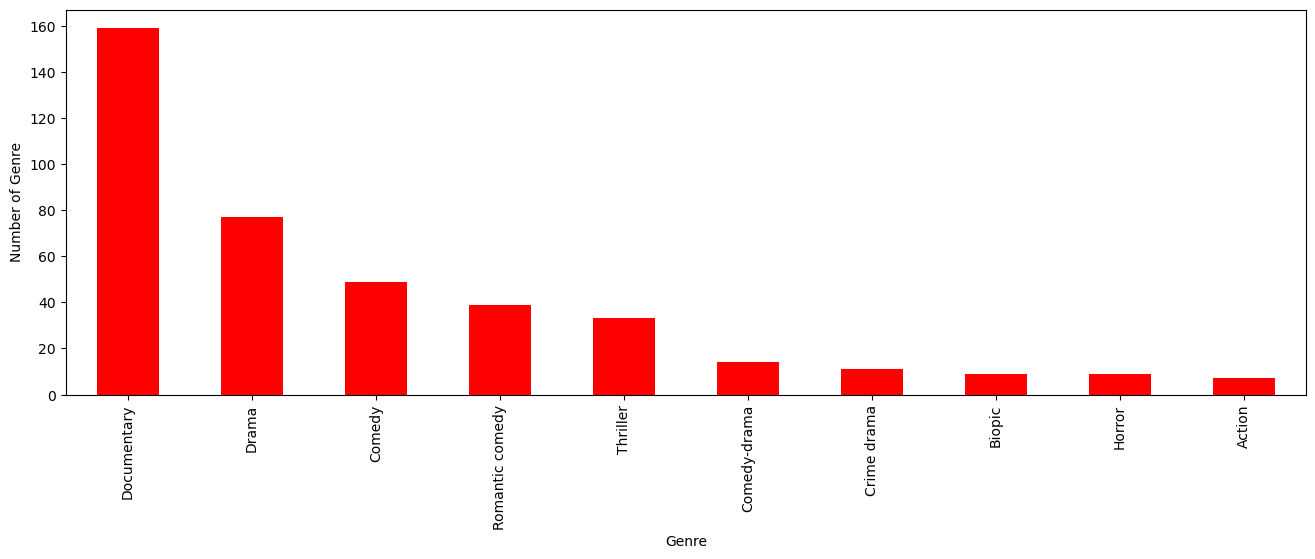
ds['Genre'].value\_counts().head(10).plot(kind='bar', color='red')

plt.xlabel('Genre')

plt.ylabel('Number of Genre')

plt.xticks(rotation=90)

plt.show(block=True)



insights: the most popular movies from genre is documentary

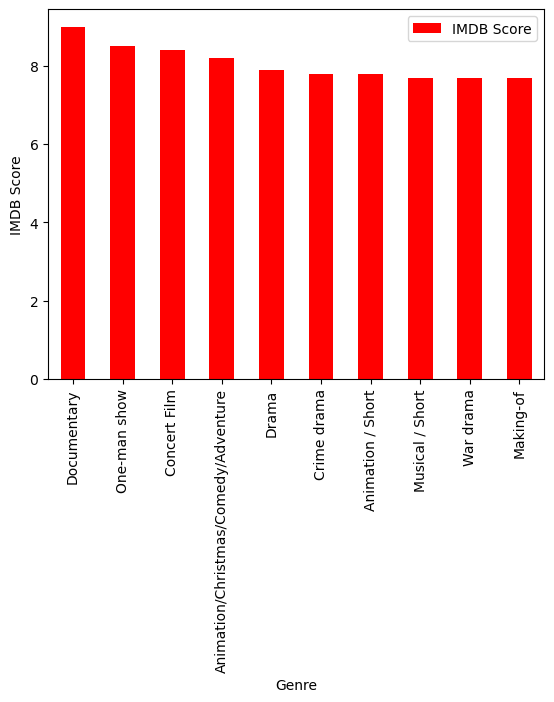
In [15]:

ds[['Genre', 'IMDB Score']].sort\_values('IMDB Score', ascending=False).drop\_duplicates('Genre').head(10).plot(x='Genre', y='IMDB Score', kind='bar', color='red')

plt.xlabel('Genre')

plt.ylabel('IMDB Score')

plt.show(block=True)



In [16]:

ds['Language'].value\_counts()

Out[16]:

English 401

Hindi 33

Spanish 31

French 20

Italian 14

Portuguese 12

Indonesian 9

Japanese 6

Korean 6

German 5

Turkish 5

English/Spanish 5

Polish 3

Dutch 3

Marathi 3

English/Hindi 2

Thai 2

English/Mandarin 2

English/Japanese 2

Filipino 2

English/Russian 1

Bengali 1

English/Arabic 1

English/Korean 1

Spanish/English 1

Tamil 1

English/Akan 1

Khmer/English/French 1

Swedish 1

Georgian 1

Thia/English 1

English/Taiwanese/Mandarin 1

English/Swedish 1

Spanish/Catalan 1

Spanish/Basque 1

Norwegian 1

Malay 1

English/Ukranian/Russian 1

Name: Language, dtype: int64

In [17]:

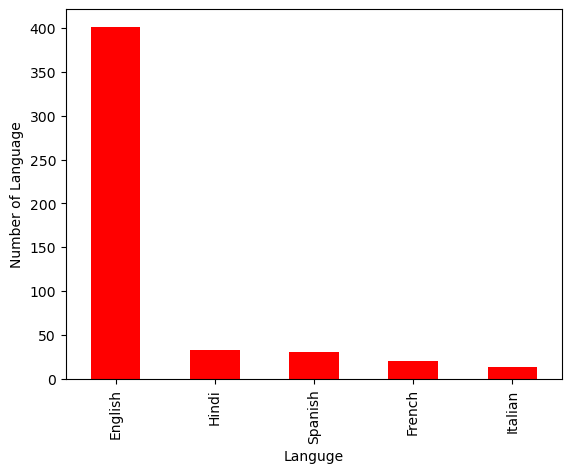
ds\_lang = ds['Language'].value\_counts()

ds\_lang.head(5).plot(kind='bar', color='red')

plt.xlabel('Languge')

plt.ylabel('Number of Language')

plt.show(block=True)



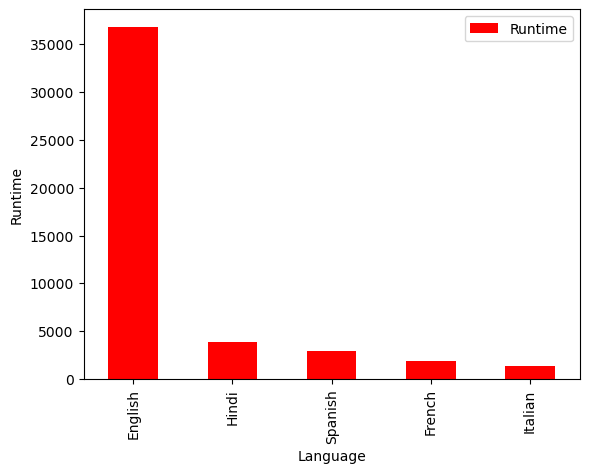
In [18]:

ds.groupby('Language').agg({'Runtime': 'sum'}).sort\_values('Runtime', ascending=False).head(5).plot(kind='bar',color='red')

plt.xlabel('Language')

plt.ylabel('Runtime')

plt.show(block=True)



In [19]:

ds\_english = ds[ds['Language'] == 'English'].sort\_values('IMDB Score', ascending=False)

ds\_english.head()

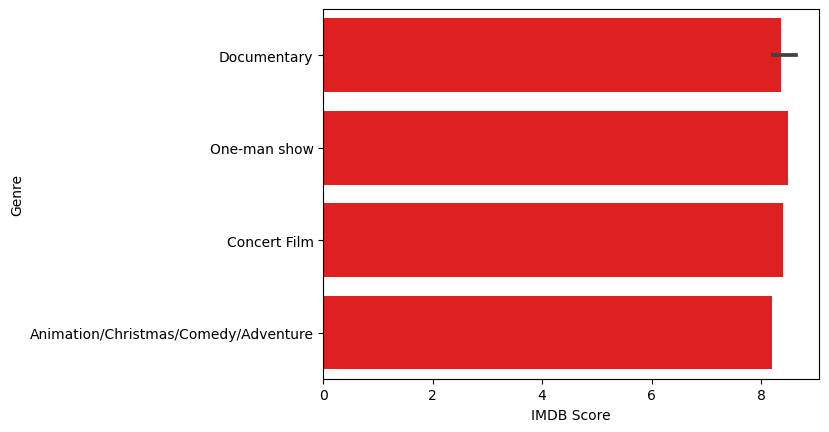
Out[19]:

|  | Title | Genre | Premiere | Runtime | IMDB Score | Language |
| --- | --- | --- | --- | --- | --- | --- |
| 583 | David Attenborough: A Life on Our Planet | Documentary | October 4, 2020 | 83 | 9.0+ | English |
| 581 | Springsteen on Broadway | One-man show | December 16, 2018 | 153 | 8.5 | English |
| 579 | Taylor Swift: Reputation Stadium Tour | Concert Film | December 31, 2018 | 125 | 8.4 | English |
| 578 | Ben Platt: Live from Radio City Music Hall | Concert Film | May 20, 2020 | 85 | 8.4 | English |
| 577 | Dancing with the Birds | Documentary | October 23, 2019 | 51 | 8.3 | English |

In [20]:

sns.barplot(y=ds\_english['Genre'].head(10), x=ds\_english['IMDB Score'], color='red')

plt.show(block=True)

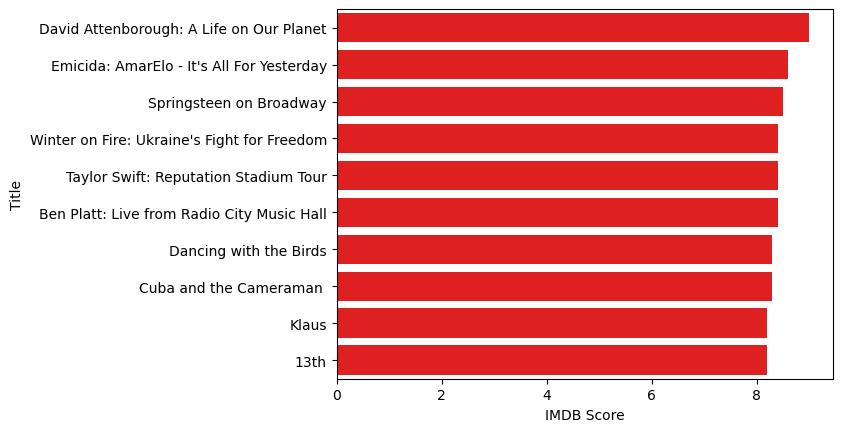


In [21]:

ds\_movie = ds[['Title', 'IMDB Score']].sort\_values('IMDB Score', ascending=False).head(10)

sns.barplot(y='Title', x='IMDB Score', data=ds\_movie, color='red')

plt.show(block=True)



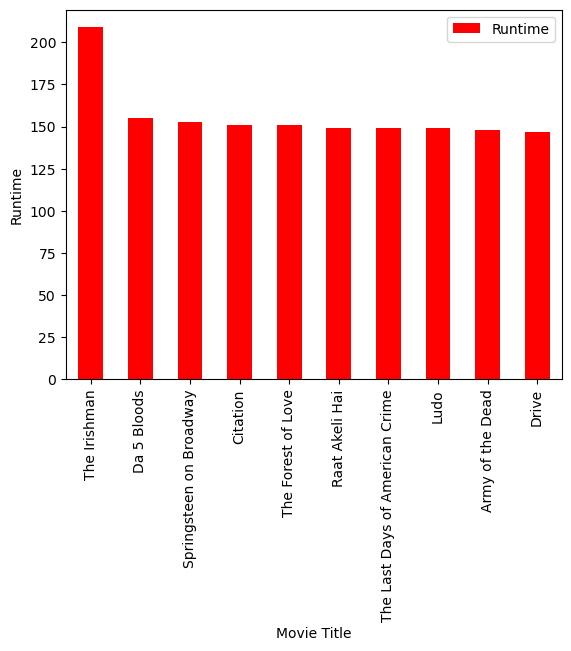
In [22]:

ds[['Title', 'Runtime']].sort\_values('Runtime', ascending=False).head(10).plot(x='Title', y='Runtime', kind='bar', color='red')

plt.xlabel('Movie Title')

plt.ylabel('Runtime')

plt.show(block=True)



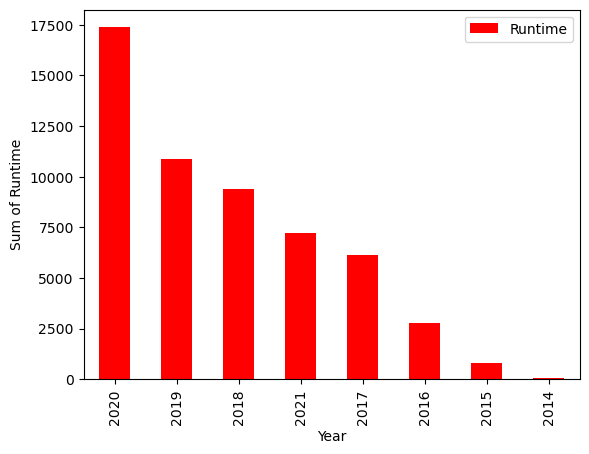
In [23]:

ds\_date.groupby('Year').agg({'Runtime': 'sum'}).sort\_values('Runtime', ascending=False).plot(kind='bar', color='red')

plt.xlabel('Year')

plt.ylabel('Sum of Runtime')

plt.show(block=True)



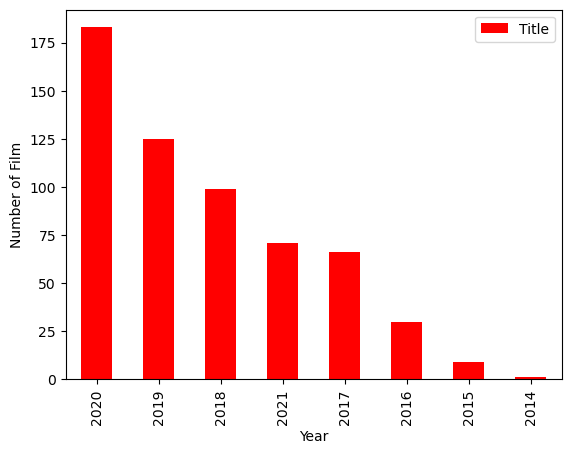
In [24]:

ds\_date.groupby('Year').agg({'Title': 'count'}).sort\_values('Title', ascending=False).plot(kind='bar', color='red')

plt.xlabel('Year')

plt.ylabel('Number of Film')

plt.show(block=True)



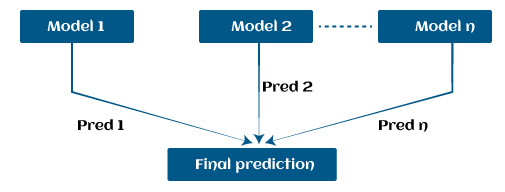
## Steps in Boosting Algorithms:

There are a few important steps in boosting the algorithm as follows:

* Consider a dataset having different data points and initialize it.
* Now, give equal weight to each of the data points.
* Assume this weight as an input for the model.
* Identify the data points that are incorrectly classified.
* Increase the weight for data points in step 4.
* If you get appropriate output then terminate this process else follow steps 2 and 3 again.

### Example:

Let's suppose, we have three different models with their predictions and they work in completely different ways. For example, the linear regression model shows a linear relationship in data while the decision tree model attempts to capture the non-linearity in the data as shown below image.



Further, instead of using these models separately to predict the outcome if we use them in form of series or combination, then we get a resulting model with correct information than all base models. In other words, instead of using each model's individual prediction, if we use average prediction from these models then we would be able to capture more information from the data. It is referred to as ensemble learning and boosting is also based on ensemble methods in machine learning.

**Boosting Algorithms in Machine Learning**

There are primarily 4 boosting algorithms in machine learning. These are as follows:

* Gradient Boosting Machine (GBM)
* Extreme Gradient Boosting Machine (XGBM)
* Light GBM
* CatBoost

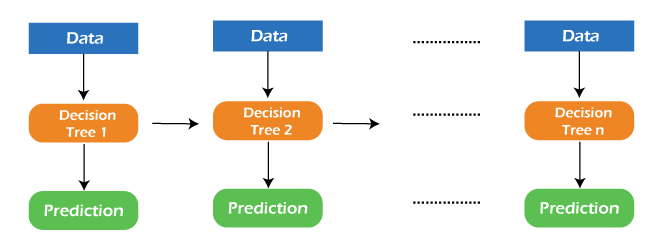
**What is GBM in Machine Learning?**

Gradient Boosting Machine (GBM) is one of the most popular forward learning ensemble methods in machine learning. It is a powerful technique for building predictive models for regression and classification tasks.

GBM helps us to get a predictive model in form of an ensemble of weak prediction models such as decision trees. Whenever a decision tree performs as a weak learner then the resulting algorithm is called gradient-boosted trees.

It enables us to combine the predictions from various learner models and build a final predictive model having the correct prediction.

But here one question may arise if we are applying the same algorithm then how multiple decision trees can give better predictions than a single decision tree? Moreover, how does each decision tree capture different information from the same data?



So, the answer to these questions is that a different subset of features is taken by the nodes of each decision tree to select the best split. It means, that each tree behaves differently, and hence captures different signals from the same data.

## How do GBM works?

Generally, most supervised learning algorithms are based on a single predictive model such as linear regression, penalized regression model, decision trees, etc. But there are some supervised algorithms in ML that depend on a combination of various models together through the ensemble. In other words, when multiple base models contribute their predictions, an average of all predictions is adapted by boosting algorithms.

Gradient boosting machines consist 3 elements as follows:

* Loss function
* Weak learners
* Additive model

Let's understand these three elements in detail.

## 1. Loss function:

Although, there is a big family of Loss functions in machine learning that can be used depending on the type of tasks being solved. The use of the loss function is estimated by the demand of specific characteristics of the conditional distribution such as robustness. While using a loss function in our task, we must specify the loss function and the function to calculate the corresponding negative gradient. Once, we get these two functions, they can be implemented into gradient boosting machines easily. However, there are several loss functions have been already proposed for GBM algorithms.

### Classification of loss function:

Based on the type of response variable y, loss function can be classified into different types as follows:

1. **Continuous response, y ∈ R:**
   * Gaussian L2 loss function
   * Laplace L1 loss function
   * Huber loss function, δ specified
   * Quantile loss function, α specified
2. **Categorical response, y ∈ {0, 1}:**
   * Binomial loss function
   * Adaboost loss function
3. **Other families of response variables:**
   * Loss functions for survival models
   * Loss functions count data

## 2. Weak Learner:

## Weak learners are the base learner models that learn from past errors and help in building a strong predictive model design for boosting algorithms in machine learning. Generally, decision trees work as a weak learners in boosting algorithms.

Weak learners are the base learner models that learn from past errors and help in building a strong predictive model design for boosting algorithms in machine learning. Generally, decision trees work as a weak learners in boosting algorithms.

Weak learners are the base learner models that learn from past errors and help in building a strong predictive model design for boosting algorithms in machine learning. Generally, decision trees work as a weak learners in boosting algorithms.

Boosting is defined as the framework that continuously works to improve the output from base models. Many gradient boosting applications allow you to "plugin" various classes of weak learners at your disposal. Hence, decision trees are most often used for weak (base) learners.

### How to train weak learners:

Machine learning uses training datasets to train base learners and based on the prediction from the previous learner, it improves the performance by focusing on the rows of the training data where the previous tree had the largest errors or residuals. E.g. shallow trees are considered weak learner to decision trees as it contains a few splits. Generally, in boosting algorithms, trees having up to 6 splits are most common.

Below is a sequence of training the weak learner to improve their performance where each tree is in the sequence with the previous tree's residuals. Further, we are introducing each new tree so that it can learn from the previous tree's errors. These are as follows:

1. Consider a data set and fit a decision tree into it.  
   **F1(x)=y**
2. Fit the next decision tree with the largest errors of the previous tree.  
   **h1(x)=y?F1(x)**
3. Add this new tree to the algorithm by adding both in steps 1 and 2.  
   **F2(x)=F1(x)+h1(x)**
4. Again fit the next decision tree with the residuals of the previous tree.  
   **h2(x)=y?F2(x)**
5. Repeat the same which we have done in step 3.  
   **F3(x)=F2(x)+h2(x)**

Continue this process until some mechanism (i.e. cross-validation) tells us to stop. The final model here is a stagewise additive model of b individual trees:

**f(x)=B∑b=1fb(x)**

Hence, trees are constructed greedily, choosing the best split points based on purity scores like Gini or minimizing the loss.

## 3. Additive Model:

The additive model is defined as adding trees to the model. Although we should not add multiple trees at a time, only a single tree must be added so that existing trees in the model are not changed. Further, we can also prefer the gradient descent method by adding trees to reduce the loss.

In the past few years, the gradient descent method was used to minimize the set of parameters such as the coefficient of the regression equation and weight in a neural network. After calculating error or loss, the weight parameter is used to minimize the error. But recently, most ML experts prefer weak learner sub-models or decision trees as a substitute for these parameters. In which, we have to add a tree in the model to reduce the error and improve the performance of that model. In this way, the prediction from the newly added tree is combined with the prediction from the existing series of trees to get a final prediction. This process continues until the loss reaches an acceptable level or is no longer improvement required.

This method is also known as functional gradient descent or gradient descent with functions.

## EXTREME GRADIENT BOOSTING MACHINE (XGBM)

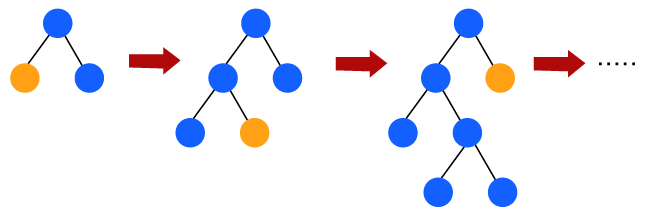
XGBM is the latest version of gradient boosting machines which also works very similar to GBM. In XGBM, trees are added sequentially (one at a time) that learn from the errors of previous trees and improve them. Although, XGBM and GBM algorithms are similar in look and feel but still there are a few differences between them as follows:

* XGBM uses various regularization techniques to reduce under-fitting or over-fitting of the model which also increases model performance more than gradient boosting machines.
* XGBM follows parallel processing of each node, while GBM does not which makes it more rapid than gradient boosting machines.
* XGBM helps us to get rid of the imputation of missing values because by default the model takes care of it. It learns on its own whether these values should be in the right or left node.

## Light Gradient Boosting Machines (Light GBM)

Light GBM is a more upgraded version of the Gradient boosting machine due to its efficiency and fast speed. Unlike GBM and XGBM, it can handle a huge amount of data without any complexity. On the other hand, it is not suitable for those data points that are lesser in number.

Instead of level-wise growth, Light GBM prefers leaf-wise growth of the nodes of the tree. Further, in light GBM, the primary node is split into two secondary nodes and later it chooses one secondary node to be split. This split of a secondary node depends upon which between two nodes has a higher loss.



Hence, due to leaf-wise split, Light Gradient Boosting Machine (LGBM) algorithm is always preferred over others where a large amount of data is given.

## CATBOOST

The catboost algorithm is primarily used to handle the categorical features in a dataset. Although GBM, XGBM, and Light GBM algorithms are suitable for numeric data sets, Catboost is designed to handle categorical variables into numeric data. Hence, catboost algorithm consists of an essential preprocessing step to convert categorical features into numerical variables which are not present in any other algorithm.

## Advantages of Boosting Algorithms:

* Boosting algorithms follow ensemble learning which enables a model to give a more accurate prediction that cannot be trumped.
* Boosting algorithms are much more flexible than other algorithms as can optimize different loss functions and provides several hyperparameter tuning options.
* It does not require data pre-processing because it is suitable for both numeric as well as categorical variables.
* It does not require imputation of missing values in the dataset, it handles missing data automatically.

## Disadvantages of Boosting Algorithms:

Below are a few disadvantages of boosting algorithms:

* Boosting algorithms may cause overfitting as well as overemphasizing the outliers.
* Gradient boosting algorithm continuously focuses to minimize the errors and requires multiple trees hence, it is computationally expensive.
* It is a time-consuming and memory exhaustive algorithm.
* Less interpretative in nature, although this is easily addressed with various tools.

# Data loading

These loading utilites can be combined with [preprocessing layers](https://keras.io/guides/preprocessing_layers/) to futher transform your input dataset before training.

Here's a quick example: let's say you have 10 folders, each containing 10,000 images from a different category, and you want to train a classifier that maps an image to its category.

Your training data folder would look like this:

training\_data/

...class\_a/

......a\_image\_1.jpg

......a\_image\_2.jpg

...class\_b/

......b\_image\_1.jpg

......b\_image\_2.jpg

etc.

You may also have a validation data folder validation\_data/ structured in the same way.

You could simply do:

from tensorflow import keras

train\_ds = keras.utils.image\_dataset\_from\_directory(

directory='training\_data/',

labels='inferred',

label\_mode='categorical',

batch\_size=32,

image\_size=(256, 256))

validation\_ds = keras.utils.image\_dataset\_from\_directory(

directory='validation\_data/',

labels='inferred',

label\_mode='categorical',

batch\_size=32,

image\_size=(256, 256))

model = keras.applications.Xception(

weights=None, input\_shape=(256, 256, 3), classes=10)

model.compile(optimizer='rmsprop', loss='categorical\_crossentropy')

model.fit(train\_ds, epochs=10, validation\_data=validation\_ds)

**Data Preprocessing**

Data preprocessing transforms the data into a format that is more easily and effectively processed in data mining, machine learning and other data science tasks. The techniques are generally used at the earliest stages of the [machine learning](https://www.techtarget.com/searchenterpriseai/definition/machine-learning-ML) and AI development pipeline to ensure accurate results.

There are several different tools and methods used for preprocessing data, including the following:

* sampling, which selects a representative subset from a large population of data;
* transformation, which manipulates raw data to produce a single input;
* denoising, which removes [noise](https://www.techtarget.com/whatis/definition/noise) from data;
* imputation, which synthesizes statistically relevant data for missing values;
* [normalization](https://searchsqlserver.techtarget.com/definition/normalization), which organizes data for more efficient access; and
* feature extraction, which pulls out a relevant feature subset that is significant in a particular context.

These tools and methods can be used on a variety of data sources, including data stored in files or databases and streaming data.

### Why is data preprocessing important?

Virtually any type of data analysis, data science or AI development requires some type of data preprocessing to provide reliable, precise and robust results for enterprise applications.

Real-world data is messy and is often created, processed and stored by a variety of humans, business processes and applications. As a result, a data set may be missing individual fields, contain manual input errors, or have duplicate data or different names to describe the same thing. Humans can often identify and rectify these problems in the data they use in the line of business, but [data used to train machine learning](https://www.techtarget.com/searchbusinessanalytics/feature/Data-preparation-in-machine-learning-6-key-steps) or deep learning algorithms needs to be automatically preprocessed.

### What are the key steps in data preprocessing?

The steps used in data preprocessing include the following:

**1. Data profiling**. Data profiling is the process of examining, analyzing and reviewing data to collect statistics about its quality. It starts with a survey of existing data and its characteristics. Data scientists identify data sets that are pertinent to the problem at hand, inventory its significant attributes, and form a hypothesis of features that might be relevant for the proposed analytics or machine learning task. They also relate data sources to the relevant business concepts and consider which preprocessing libraries could be used.

**2. Data cleansing**. The aim here is to find the easiest way to rectify quality issues, such as eliminating bad data, filling in missing data or otherwise ensuring the raw data is suitable for feature engineering.

**3. Data reduction.**Raw data sets often include redundant data that arise from characterizing phenomena in different ways or data that is not relevant to a particular ML, AI or analytics task. Data reduction uses techniques like principal component analysis to transform the raw data into a simpler form suitable for particular use cases.

**4. Data transformation**. Here, data scientists think about how different aspects of the data need to be organized to make the most sense for the goal. This could include things like structuring unstructured data, combining salient variables when it makes sense or identifying important ranges to focus on.

**5. Data enrichment**. In this step, data scientists apply the various feature engineering libraries to the data to effect the desired transformations. The result should be a data set organized to achieve the optimal balance between the training time for a new model and the required compute.

**6. Data validation**. At this stage, the data is split into two sets. The first set is used to train a machine learning or deep learning model. The second set is the testing data that is used to gauge the accuracy and robustness of the resulting model. This second step helps identify any problems in the hypothesis used in the cleaning and feature engineering of the data. If the data scientists are satisfied with the results, they can push the preprocessing task to a data engineer who figures out how to scale it for production. If not, the data scientists can go back and make changes to the way they implemented the data cleansing and feature engineering steps.

### Data preprocessing techniques

There are two main categories of preprocessing -- data cleansing and feature engineering. Each includes a variety of techniques, as detailed below.

### Data cleansing

Techniques for cleaning up messy data include the following:

**Identify and sort out missing data.**There are a variety of reasons a data set might be missing individual fields of data. Data scientists need to decide whether it is better to discard records with missing fields, ignore them or fill them in with a probable value. For example, in an [IoT](https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT) application that records temperature, adding in a missing average temperature between the previous and subsequent record might be a safe fix.

**Reduce**[**noisy data**](https://www.techtarget.com/searchbusinessanalytics/definition/noisy-data)**.** Real-world data is often noisy, which can distort an analytic or AI model. For example, a temperature sensor that consistently reported a temperature of 75 degrees Fahrenheit might erroneously report a temperature as 250 degrees. A variety of statistical approaches can be used to reduce the noise, including binning, regression and clustering.

**Identify and remove duplicates.** When two records seem to repeat, an algorithm needs to determine if the same measurement was recorded twice, or the records represent different events. In some cases, there may be slight differences in a record because one field was recorded incorrectly. In other cases, records that seem to be duplicates might indeed be different, as in a father and son with the same name who are living in the same house but should be represented as separate individuals. Techniques for identifying and removing or joining duplicates can help to automatically address these types of problems.

### Feature engineering

Feature engineering, as noted, involves techniques used by data scientists to organize the data in ways that make it more efficient to train [data models](https://www.techtarget.com/searchdatamanagement/definition/data-modeling) and run inferences against them. These techniques include the following:

**Feature scaling or normalization.**Often, multiple variables change over different scales, or one will change linearly while another will change exponentially. For example, salary might be measured in thousands of dollars, while age is represented in double digits. Scaling helps to transform the data in a way that makes it easier for algorithms to tease apart a meaningful relationship between variables.

**Data reduction.**Data scientists often need to combine a variety of [data sources to create a new AI or analytics model](https://www.techtarget.com/searchbusinessanalytics/feature/6-data-preparation-best-practices-for-analytics-applications). Some of the variables may not be correlated with a given outcome and can be safely discarded. Other variables might be relevant, but only in terms of relationship -- such as the ratio of debt to credit in the case of a model predicting the likelihood of a loan repayment; they may be combined into a single variable. Techniques like principal component analysis play a key role in reducing the number of dimensions in the training data set into a more efficient representation.

**Discretization.**It's often useful to lump raw numbers into discrete intervals. For example, income might be broken into five ranges that are representative of people who typically apply for a given type of loan. This can reduce the overhead of training a model or running inferences against it.

**Feature encoding**. Another aspect of feature engineering involves organizing unstructured data into a structured format. Unstructured data formats can include text, audio and video. For example, the process of developing natural language processing algorithms typically starts by using data transformation algorithms like Word2vec to translate words into numerical vectors. This makes it easy to represent to the algorithm that words like "mail" and "parcel" are similar, while a word like "house" is completely different. Similarly, a [facial recognition](https://www.techtarget.com/searchenterpriseai/definition/facial-recognition) algorithm might reencode raw pixel data into vectors representing the distances between parts of the face.

### How is data preprocessing used?

Data preprocessing plays a key role in earlier stages of machine learning and AI application development, as noted earlier. In an [AI](https://www.techtarget.com/searchenterpriseai/definition/AI-Artificial-Intelligence) context, data preprocessing is used to improve the way data is cleansed, transformed and structured to improve the accuracy of a new model, while reducing the amount of compute required.

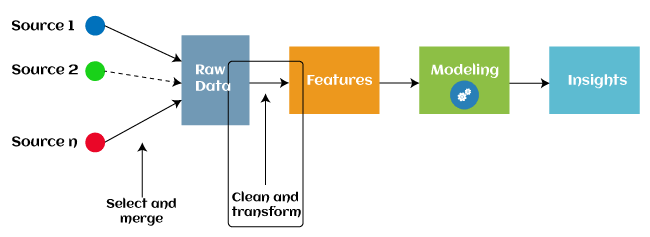
A good data preprocessing pipeline can create reusable components that make it easier to test out various ideas for streamlining business processes or improving customer satisfaction. For example, preprocessing can improve the way data is organized for a recommendation engine by improving the age ranges used for categorizing customers.

Preprocessing can also simplify the work of creating and modifying data for more accurate and targeted business intelligence insights. For example, customers of different sizes, categories or regions may exhibit different behaviors across regions. Preprocessing the data into the appropriate forms could help BI teams weave these insights into BI dashboards.

In a customer relationship management ([CRM](https://www.techtarget.com/searchcustomerexperience/definition/CRM-customer-relationship-management)) context, data preprocessing is a component of web mining. Web usage logs may be preprocessed to extract meaningful sets of data called user transactions, which consist of groups of URL references. User sessions may be tracked to identify the user, the websites requested and their order, and the length of time spent on each one. Once these have been pulled out of the raw data, they yield more useful information that can be applied, for example, to consumer research, marketing or [personalization](https://www.techtarget.com/searchcustomerexperience/definition/personalization).

**What is Feature Engineering?**

**Feature engineering is the pre-processing step of machine learning, which extracts features from raw data**. It helps to represent an underlying problem to predictive models in a better way, which as a result, improve the accuracy of the model for unseen data. The predictive model contains predictor variables and an outcome variable, and while the feature engineering process selects the most useful predictor variables for the model.



Since 2016, automated feature engineering is also used in different machine learning software that helps in automatically extracting features from raw data. Feature engineering in ML contains mainly four processes: **Feature Creation, Transformations, Feature Extraction, and Feature Selection.**

**These processes are described as below:**

**feature Creation**: Feature creation is finding the most useful variables to be used in a predictive model. The process is subjective, and it requires human creativity and intervention. The new features are created by mixing existing features using addition, subtraction, and ration, and these new features have great flexibility.

**Transformations**: The transformation step of feature engineering involves adjusting the predictor variable to improve the accuracy and performance of the model. For example, it ensures that the model is flexible to take input of the variety of data; it ensures that all the variables are on the same scale, making the model easier to understand. It improves the model's accuracy and ensures that all the features are within the acceptable range to avoid any computational error.

## Feature Extraction: Feature extraction is an automated feature engineering process that generates new variables by extracting them from the raw data. The main aim of this step is to reduce the volume of data so that it can be easily used and managed for data modelling. Feature extraction methods include cluster analysis, text analytics, edge detection algorithms, and principal components analysis (PCA).

**Feature Selection:** While developing the machine learning model, only a few variables in the dataset are useful for building the model, and the rest features are either redundant or irrelevant. If we input the dataset with all these redundant and irrelevant features, it may negatively impact and reduce the overall performance and accuracy of the model. Hence it is very important to identify and select the most appropriate features from the data and remove the irrelevant or less important features, which is done with the help of feature selection in machine learning. ***"Feature selection is a way of selecting the subset of the most relevant features from the original features set by removing the redundant, irrelevant, or noisy features."***

**Steps in Feature Engineering**

The steps for feature engineering vary per different Ml engineers and data scientists. Some of the common steps that are involved in most machine-learning algorithms are:

### 1. Data Cleansing

Data cleansing (also known as data cleaning or data scrubbing) involves identifying and removing or correcting any errors or inconsistencies in the dataset. This step is important to ensure that the data is accurate and reliable.

### 2. Data Transformation

### Data transformation involves converting and scaling variables in the dataset to make them more useful for machine learning. This can include techniques like normalization, standardization, and log transformation.

### 3. Feature Extraction

Feature extraction involves creating new features from the existing variables in the dataset. This can include techniques like principal component analysis (PCA), text parsing, and image processing.

### 4. Feature Selection

Feature selection involves selecting the most relevant features from the dataset for use in machine learning. This can include techniques like correlation analysis, mutual information, and stepwise regression.

### 5. Feature Iteration

Feature iteration involves refining and improving the features based on the performance of the machine learning model. This can include techniques like adding new features, removing redundant features and transforming features in different ways.

Overall, the goal of feature engineering is to create a set of informative and relevant features that can be used to train a machine learning model and improve its accuracy and performance. The specific steps involved in the process may vary depending on the type of data and the specific machine-learning problem at hand.

 useful for reducing the impact of small variations in the data and making it easier to analyze. Binning is the process of grouping continuous features into discrete bins. This can help simplify the feature and reduce noise in the data. Binning can be performed using equal width or equal frequency intervals.

### 6. Feature Split

Feature split is the process of splitting a single variable into multiple variables. This is often done when a variable contains multiple pieces of information that can be more easily analyzed separately. Feature split involves splitting a feature into multiple features. For example, a feature representing a date can be split into year, month, and day features. This can help capture more information about the data and improve the performance of machine learning models.

**What is model training?**

A training model is a dataset that is used to train an ML algorithm. It consists of the sample output data and the corresponding sets of input data that have an influence on the output. The training model is used to run the input data through the algorithm to correlate the processed output against the sample output. The result from this correlation is used to modify the model.

This iterative process is called “model fitting”. The accuracy of the training dataset or the validation dataset is critical for the precision of the model.

Model training in machine language is the process of feeding an ML algorithm with data to help identify and learn good values for all attributes involved. There are several types of machine learning models, of which the most common ones are supervised and unsupervised learning.

Supervised learning is possible when the training data contains both the input and output values. Each set of data that has the inputs and the expected output is called a supervisory signal. The training is done based on the deviation of the processed result from the documented result when the inputs are fed into the model.

Unsupervised learning involves determining patterns in the data. Additional data is then used to fit patterns or clusters. This is also an iterative process that improves the accuracy based on the correlation to the expected patterns or clusters. There is no reference output dataset in this method.

## Why Is Model Training Important?

Model training aims to build the best mathematical representation of the relationship between data and a target (supervised) or among the data itself (unsupervised).

Metrics such as accuracy define how well the model has learned this representation, i.e. they report the model’s performance. The better the model performance, the more benefits using the model in real life will bring. These benefits could include increased revenue, reduced costs, or improved user experience.

Investing time and resources for optimal model training means having access to the right expertise and an appropriate engineering backbone setup within a production-first approach to ML. Such an investment can prove a real differentiator for business success. In fact, leading ML-driven businesses achieve 44% higher productivity and 40% better customer experience among other gains than counterparts.

## How to Train a Machine Learning Model?

The process of training ML models can be divided into four steps.

**Data Set Split for Training and Evaluation**

The training data set is used for model training, and the evaluation set for performance evaluation of the trained model. It is essential that these sets do not intersect and that data in the evaluation sets has not been seen during training in order to ensure an unbiased performance estimate.

**Algorithm Selection**

First, we should select a simpler algorithm than our model’s, or a heuristic, to use as a baseline to compare the final trained model’s performance again.

Then, it is common to select multiple algorithms for training, speed, costs, data size and type, available infrastructure, and desired unless one specific algorithm is clearly the best fit for the use case and data. The most appropriate algorithm(s) to deploy is dependent upon training and inference offline performance.

Some of the most common machine modeling techniques are:

* Linear regression, SVM, random forest, boosted trees, and neural networks\*, for supervised learning
* K-means for unsupervised learning

For deep learning, there is a follow-up phase of “model architecture development” to define the exact layers—optionally on top of pretrained networks—to be used for the final neural network model.

### Hyperparameter Tuning

Each algorithm has a set of default hyperparameters, which is unlikely to be the most performant for any use case and data. We perform hyperparameter tuning on a data subset before training the final model on the complete data set to maximize the performance from each algorithm.

We should also provide a validation set when performing [model tuning](https://www.iguazio.com/glossary/model-tuning/) for evaluation with different hyperparameter selections so as to keep the evaluation set unseen for the final model evaluation.

**Fit and Tune Models**

Now that we’ve split our dataset into training and test sets, and we’ve learned about hyperparameters and cross-validation, we’re ready fit and tune our models. Basically, all we need to do is perform the entire cross-validation loop detailed above on each **set of hyperparameter values** we’d like to try.

The high-level pseudo-code looks like this:

Pseudocode for Tuning Hyperparameters

|  |  |
| --- | --- |
|  | For each algorithm (i.e. regularized regression, random forest, etc.):  For each set of hyperparameter values to try:  Perform cross-validation using the training set.  Calculate cross-validated score. |

|  |  |
| --- | --- |
|  |  |

At the end of this process, you will have a cross-validated score for each set of hyperparameter values… for each algorithm.

For example:



Then, we’ll pick the best set of hyperparameters *within each algorithm*.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Pseudocode for Choosing Hyperparameters   |  |  | | --- | --- | |  | For each algorithm:   Keep the set of hyperparameter values with best cross-validated score.  Re-train the algorithm on the entire training set (without cross-validation). | |

It’s kinda like the Hunger Games… each algorithm sends its own “representatives” (i.e. model trained on the best set of hyperparameter values) to the final selection.

## What Is Model Evaluation?

Model evaluation in machine learning is the process of determining a model’s performance via a metrics-driven analysis. It can be performed in two ways:

* Offline: The model is evaluated after training during experimentation or continuous retraining.
* Online: The model is evaluated in production as part of model monitoring.

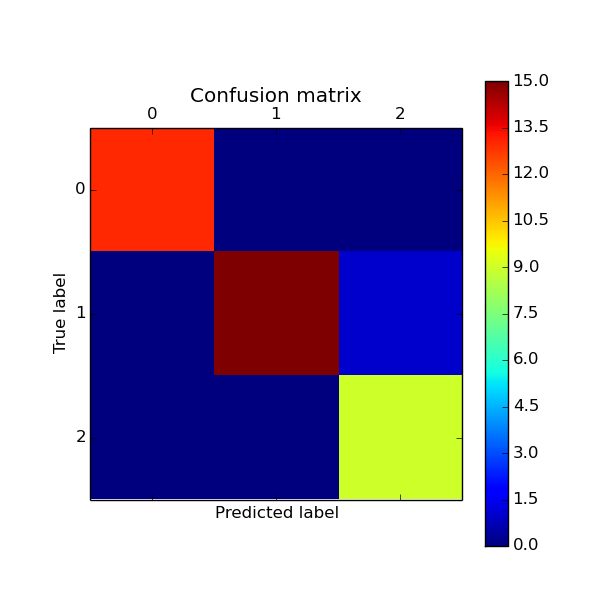
**Classification**

When analyzing classification models choosing the right metric for evaluating machine learning models is of paramount importance. Here is the list of model metrics:

* **Accuracy**is defined as the proportion of correct outcomes to total cases. Strive for a high level of precision.
* **Log loss** is a clinical outcome that shows the classifier’s advantage over a random guess. The log loss quantifies your model’s uncertainty by comparing the probability of its outputs against known values. You want to reduce log loss for the whole model.
* **The Confusion Matrix** is the relationship between the label and the categorization of the model. A confusion matrix has one axis for the expected label and another for the actual label.
* **The area under the curve (AUC)** is calculated by plotting false positives on the x-axis and true positives on the y-axis. This statistic is significant since it gives a single value that allows you to compare different types of models.
* **Precision** is defined as the ratio of correct outcomes to all positive results.
* **The recall** is the percentage of correct answers provided by the model.
* **The F1-score**is another machine learning model evaluation metric. It’s the weighted average of accuracy and recall between 0 and 1, with 1 being the optimal F-score.
* There are two critical steps in developing an ML model for a specific problem statement: training and testing. During the training phase, the models adapt from the data and forecast the final results. However, the generated model’s predictions must be correct. Because it can ensure how accurate the outcomes were to execute for the specified problem, testing is the most important step.

**Example of confusion matrix**

Example of confusion matrix usage to evaluate the quality of the output of a classifier on the iris data set. The diagonal elements represent the number of points for which the predicted label is equal to the true label, while off-diagonal elements are those that are mislabeled by the classifier. The higher the diagonal values of the confusion matrix the better, indicating many correct predictions.



**Script output**:

[[13 0 0]

[ 0 15 1]

[ 0 0 9]]

**Python source code:** [plot\_confusion\_matrix.py](https://scikit-learn.org/0.15/_downloads/plot_confusion_matrix1.py)

print(\_\_doc\_\_)

from sklearn import svm, datasets

from sklearn.cross\_validation import [train\_test\_split](https://scikit-learn.org/0.15/modules/generated/sklearn.cross_validation.train_test_split.html#sklearn.cross_validation.train_test_split)

from sklearn.metrics import [confusion\_matrix](https://scikit-learn.org/0.15/modules/generated/sklearn.metrics.confusion_matrix.html#sklearn.metrics.confusion_matrix)

import matplotlib.pyplot as plt

*# import some data to play with*

iris = datasets.load\_iris()

X = iris.data

y = iris.target

*# Split the data into a training set and a test set*

X\_train, X\_test, y\_train, y\_test = [train\_test\_split](https://scikit-learn.org/0.15/modules/generated/sklearn.cross_validation.train_test_split.html#sklearn.cross_validation.train_test_split)(X, y, random\_state=0)

*# Run classifier*

classifier = [svm.SVC](https://scikit-learn.org/0.15/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC)(kernel='linear')

y\_pred = classifier.fit(X\_train, y\_train).predict(X\_test)

*# Compute confusion matrix*

cm = [confusion\_matrix](https://scikit-learn.org/0.15/modules/generated/sklearn.metrics.confusion_matrix.html#sklearn.metrics.confusion_matrix)(y\_test, y\_pred)

print(cm)

*# Show confusion matrix in a separate window*

[plt.matshow](http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.matshow)(cm)

[plt.title](http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.title)('Confusion matrix')

[plt.colorbar](http://matplotlib.org/api/colorbar_api.html#matplotlib.colorbar)()

[plt.ylabel](http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.ylabel)('True label')

[plt.xlabel](http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.xlabel)('Predicted label')

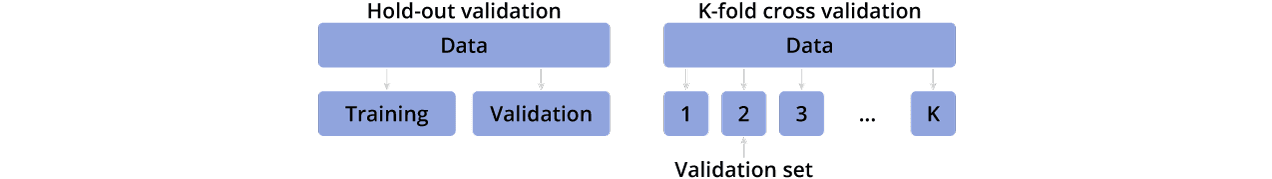
[plt.show](http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.show)()

**Model Evaluation Techniques**

We have known that the model evaluation is an Integral part in Machine Learning. Initially, the dataset is divided into two types, they are “Training dataset” and “Test dataset”. We build the machine learning model using the training dataset to see the functionality of the model. But we evaluate the designed Model using a test dataset, which consists of unseen or unknown samples of the data that are not used for training purposes. Evaluation of a model tells us how accurate the results were. If we use the training dataset for evaluation of the model, for any instance of the training data it will always show the correct predictions for the given problem with high accuracy measures, in that case our model is not adequately effective to use.

There are two methods that are used to evaluate a model performance. They are

1. Holdout
2. Cross Validation



The Holdout method is used to evaluate the model performance and uses two types of data for testing and training. The test data is used to calculate the performance of the model whereas it is trained using the training data set.  This method is used to check how well the machine learning model developed using different algorithm techniques performs on unseen samples of data. This approach is simple, flexible and fast.

Cross-validation is a procedure of dividing the whole dataset into data samples, and then evaluating the machine learning model using the other samples of data to know accuracy of the model. i.e., we train the model using a subset of data and we evaluate it with a complementary data subset. We can calculate cross validation based on the following 3 methods, namely

1. Validation
2. Leave one out cross validation (LOOCV)
3. K-Fold Cross Validation

In the method of validation, we split the given dataset into 50% of training and 50% for testing purpose. The main drawback in this method is that the remaining 50% of data that is subjected to testing may contain some crucial information that may be lost while training the model. So, this method doesn’t work properly due to high bias.

In the method of LOOCV, we train all the datasets in our model and leave a single data point for testing purpose. This method aims at exhibiting lower bias, but there are some chances that this method might fail because, the data-point that has been left out may be an outlier in the given data; and in that case we cannot produce better results with good accuracy.

K-fold cross validation is a popular method used for evaluation of a Machine Learning model. It works by splitting the data into k-parts. Each split of the data is called a fold. Here we train all the k subsets of data to the model, and then we leave out one (k-1) subset to perform evaluation on the trained model. This method results in high accuracy and produces data with less bias.

**Conclusion:**

IMDB is a popular website where users can rate and review movies. The website

Has a large user base, which makes it a valuable resource for moviegoers. However, the IMDB rating system is not perfect. It is possible for movies to be rated highly or poorly due to factors other than their quality. For example, a movie with a large marketing budget may receive more positive reviews than it deserves.

One way to improve the IMDB rating system is to use machine learning to predict the ratings of movies. Machine learning algorithms can be trained on data from past movies to learn how different factors, such as the director, cast, and genre, affect the rating of a movie. This information can then be used to predict the rating of a new movie before it is released.

Machine learning can also be used to identify bias in the IMDB rating system. For example, if a movie is rated highly by male users but poorly by female users, this may indicate that the rating system is biased against female users. Machine learning algorithms can be used to identify this type of bias and to adjust the rating system accordingly.

Overall, machine learning can be used to improve the IMDB rating system by making it more accurate and less biased. This would make the website a more valuable resource for moviegoers.

Here are some of the ways that machine learning can be used to predict IMDB scores:

* Using data from past movies. Machine learning algorithms can be trained on data from past movies to learn how different factors, such as the director, cast, and genre, affect the rating of a movie. This information can then be used to predict the rating of a new movie before it is released.
* Identifying bias in the IMDB rating system. Machine learning algorithms can be used to identify bias in the IMDB rating system. For example, if a movie is rated highly by male users but poorly by female users, this may indicate that the rating system is biased against female users. Machine learning algorithms can be used to identify this type of bias and to adjust the rating system accordingly.
* Using natural language processing to analyze reviews. Machine learning algorithms can be used to analyze reviews of movies to extract information about the movie's quality. This information can then be used to predict the rating of the movie.
* Using social media data. Machine learning algorithms can be used to analyze social media data about movies to extract information about the movie's popularity. This information can then be used to predict the rating of the movie.

Overall, machine learning can be used to improve the IMDB rating system by making it more accurate and less biased. This would make the website a more valuable resource for moviegoers.