**Predicting IMDb Scores Using Machine Learning**

**TEAM MEMBER**

**732521104054 : THIRISHA 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.

In this part you will need to understand the problem statement and create a document on what have you understood and how will you proceed ahead with solving the problem. Please think on a design and present in form of the document.

In this section you need to put your design into innovation to solve the problem. Create a doc around it and share the same for assessment.

In this section begin building your project by loading and preprocessing the dataset.

In this section continue building the project by performing different activities like feature engineering, model training, evaluation etc as per the instructions in the project.

**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

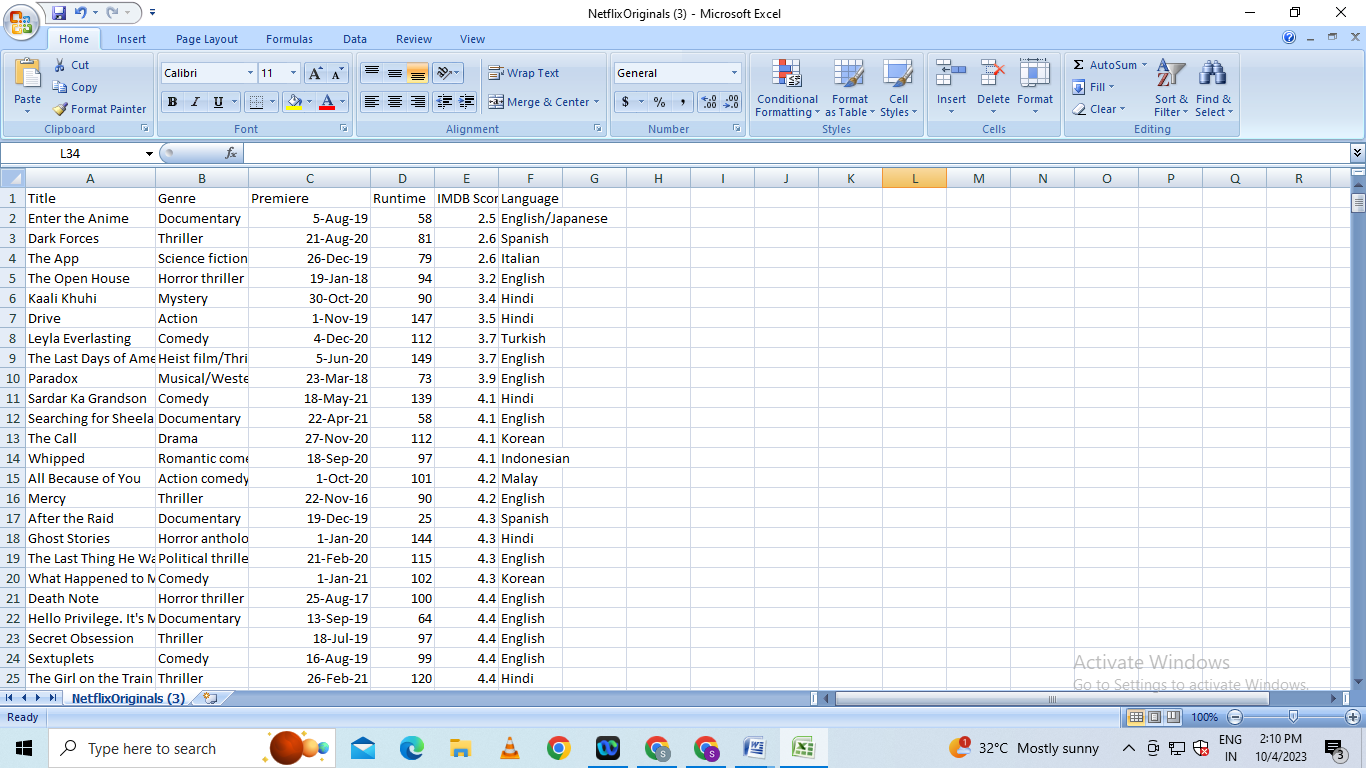
13: end if

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))

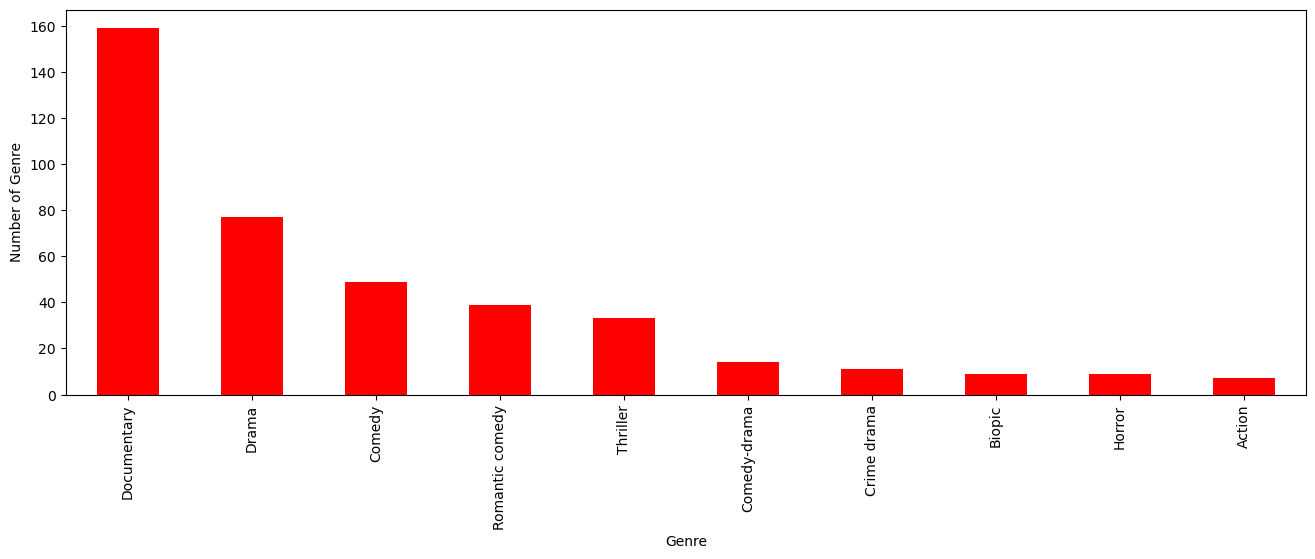
ds['Genre'].value\_counts().head(10).plot(kind='bar', color='red')

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

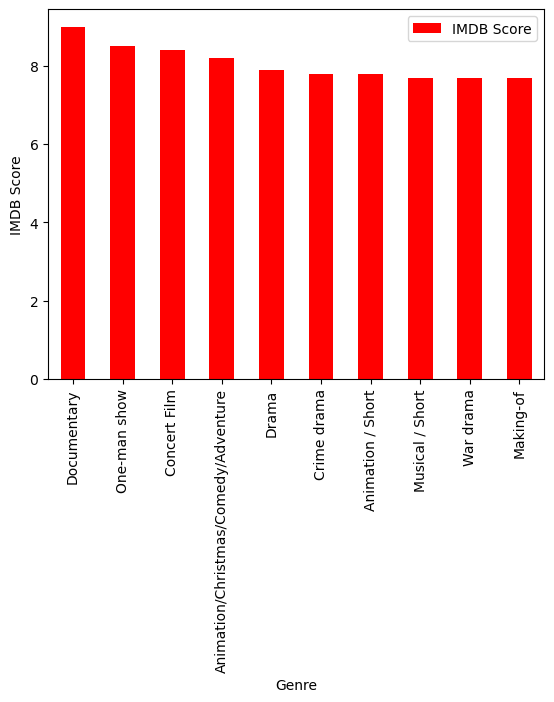
In [15]:

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]:

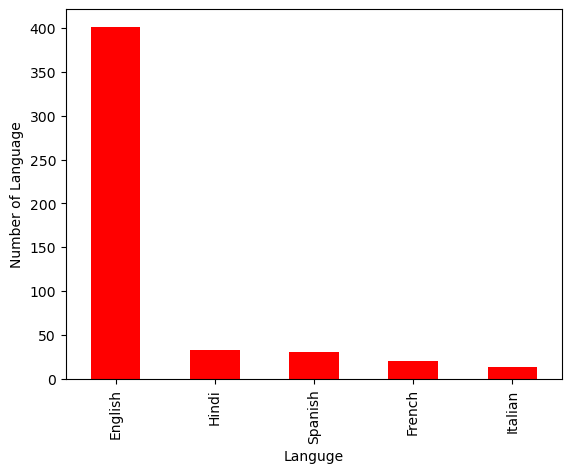
ds\_lang = ds['Language'].value\_counts()

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

plt.xlabel('Languge')

plt.ylabel('Number of Language')

plt.show(block=True)



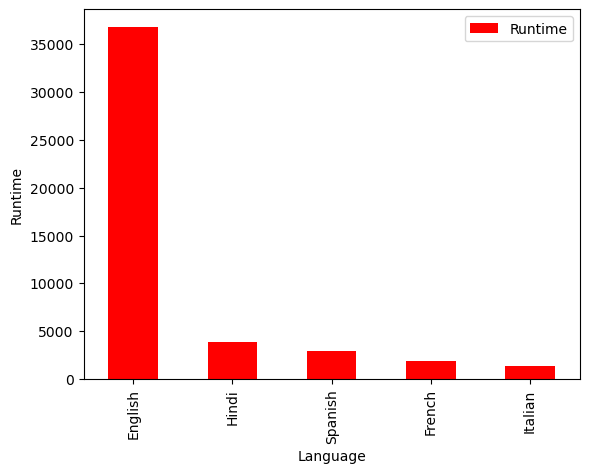
In [18]:

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()

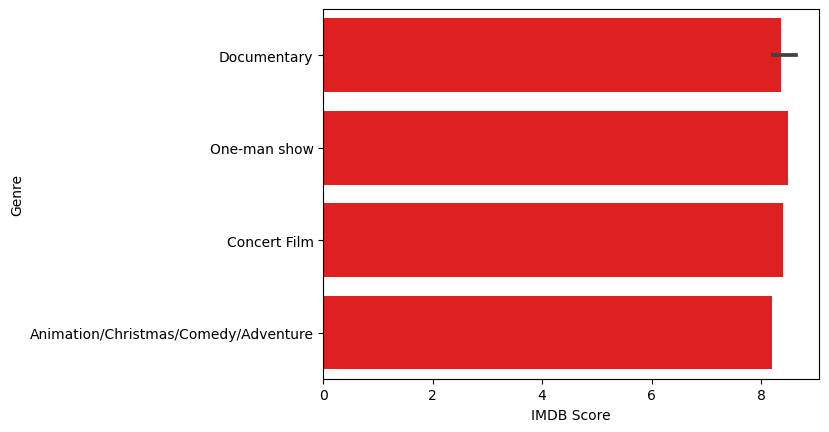
Out[19]:

|  | 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)

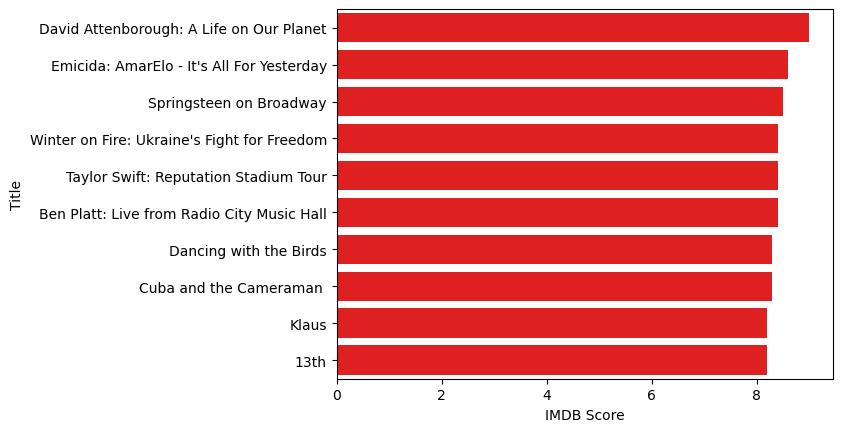


In [21]:

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)



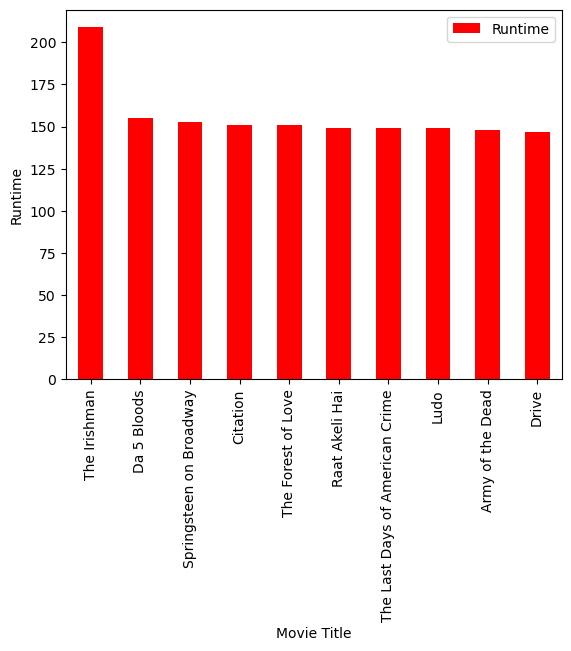
In [22]:

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)



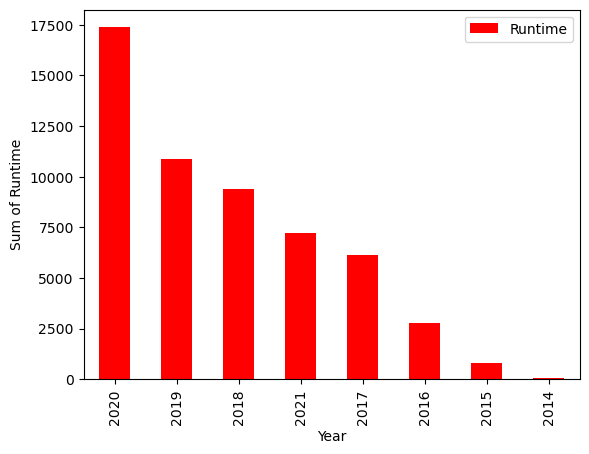
In [23]:

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)



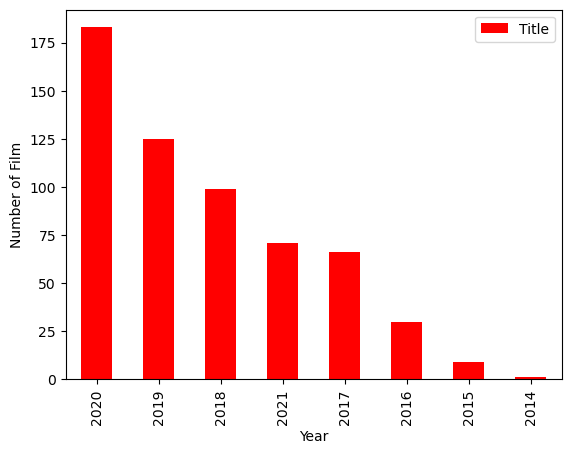
In [24]:

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)



**Model Evaluation and Selection:**

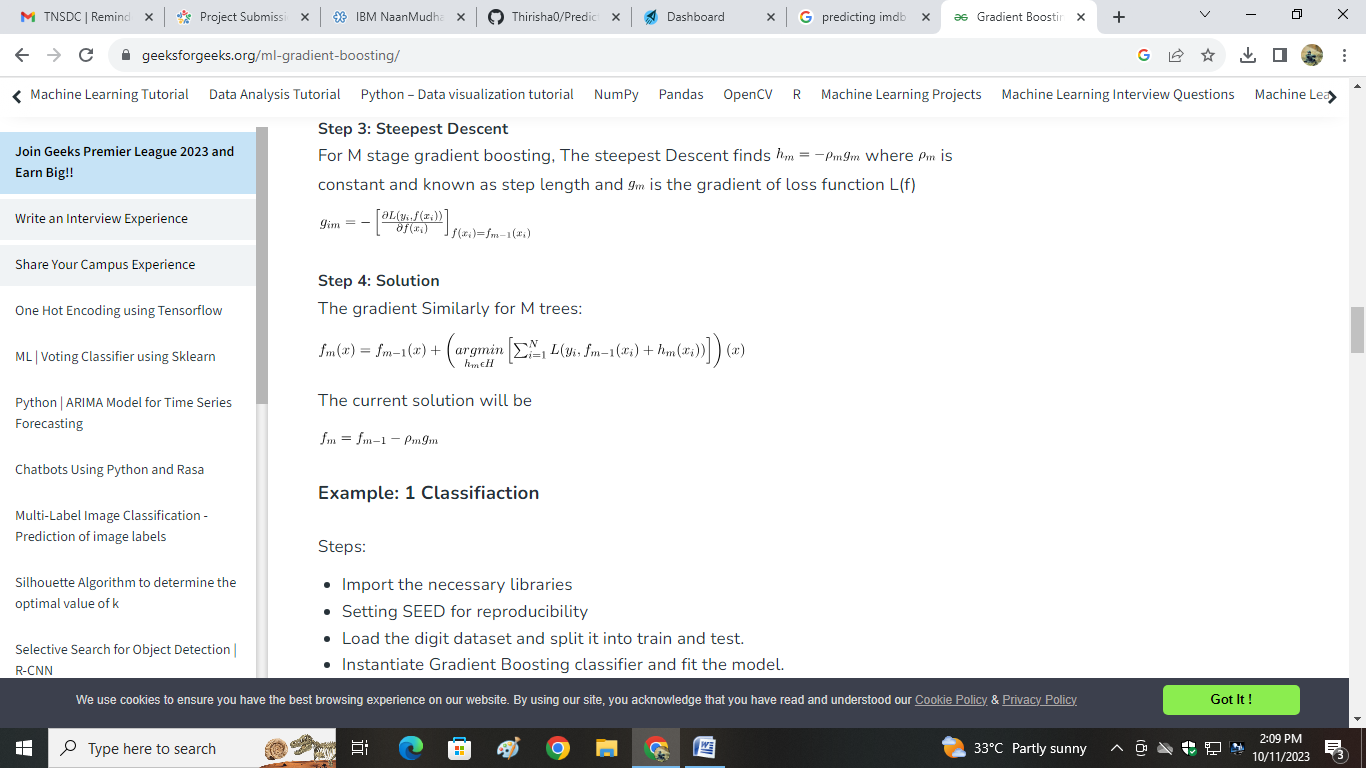
* Split the dataset into training and testing sets.
* Evaluate models using appropriate metrics (e.g., Mean Absolute Error, Mean Squared Error, R-squared) to assess their performance.
* Use cross-validation techniques to tune hyperparameters and ensure model stability. Compare the results with traditional linear regression models to highlight improvements.
* Select the best-performing model for further analysis.

**Gradient Boosting Algorithm:**

**Step 1:**

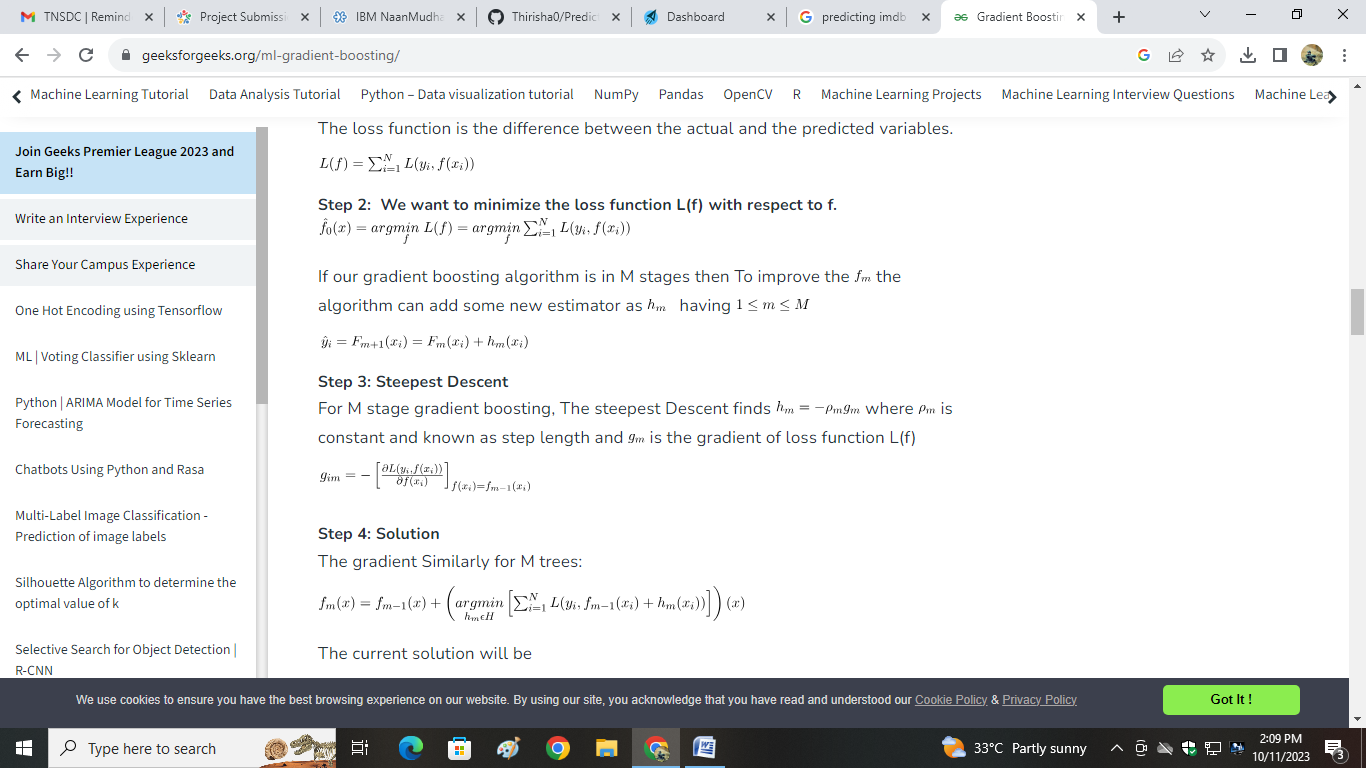
Let’s assume X, and Y are the input and target having N samples. Our goal is to learn the function f(x) that maps the input features X to the target variables y. It is boosted trees i.e the sum of trees.

The loss function is the difference between the actual and the predicted variables.

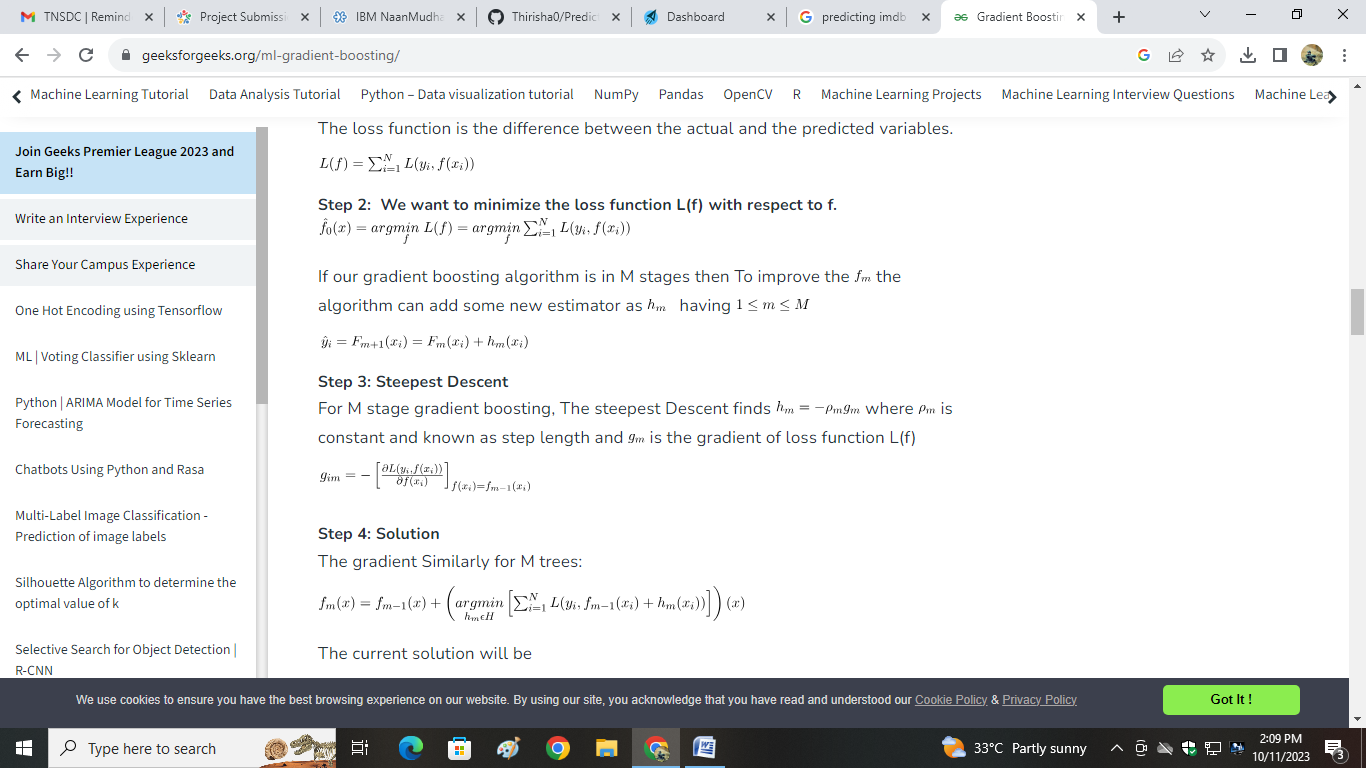


**Step 2:  We want to minimize the loss function L(f) with respect to f.**

If our gradient boosting algorithm is in M stages then To improve the  the algorithm can add some new estimator as    having

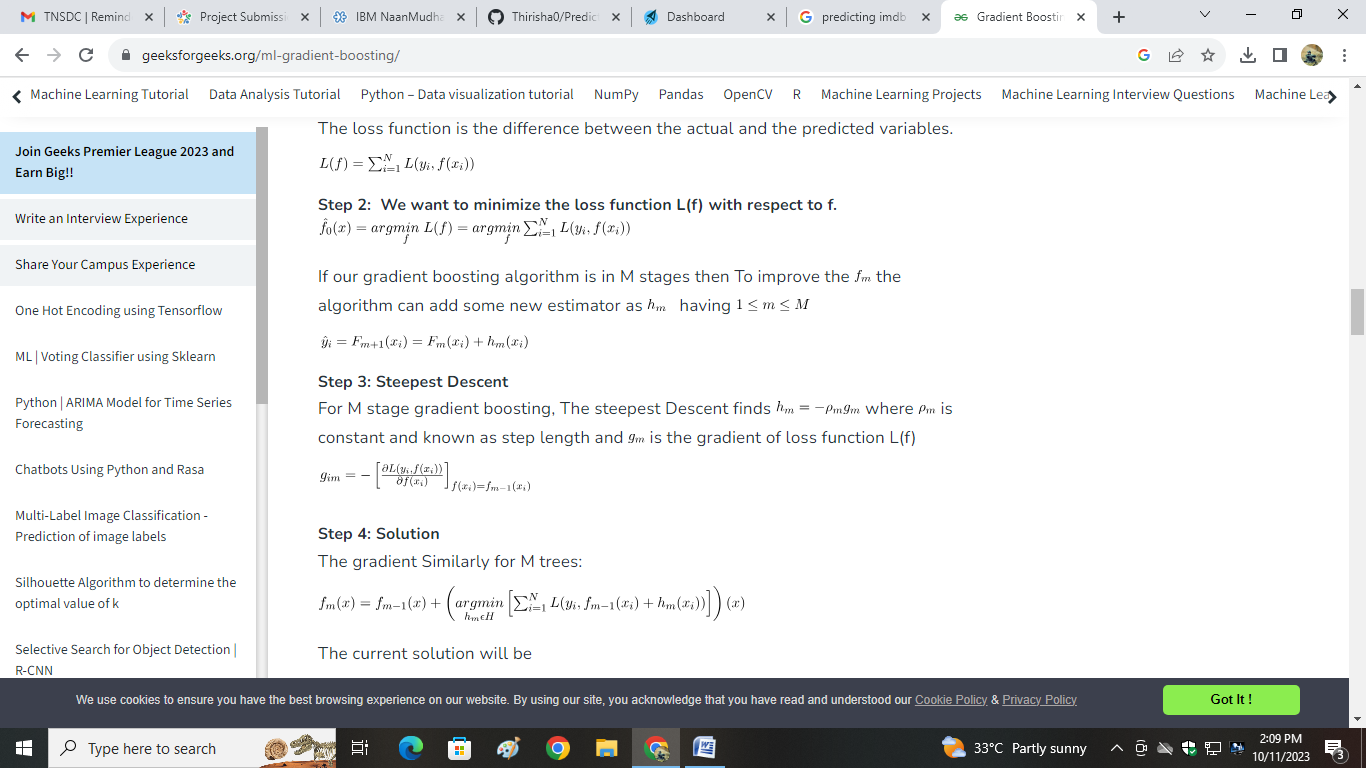


**Step 3: Steepest Descent**

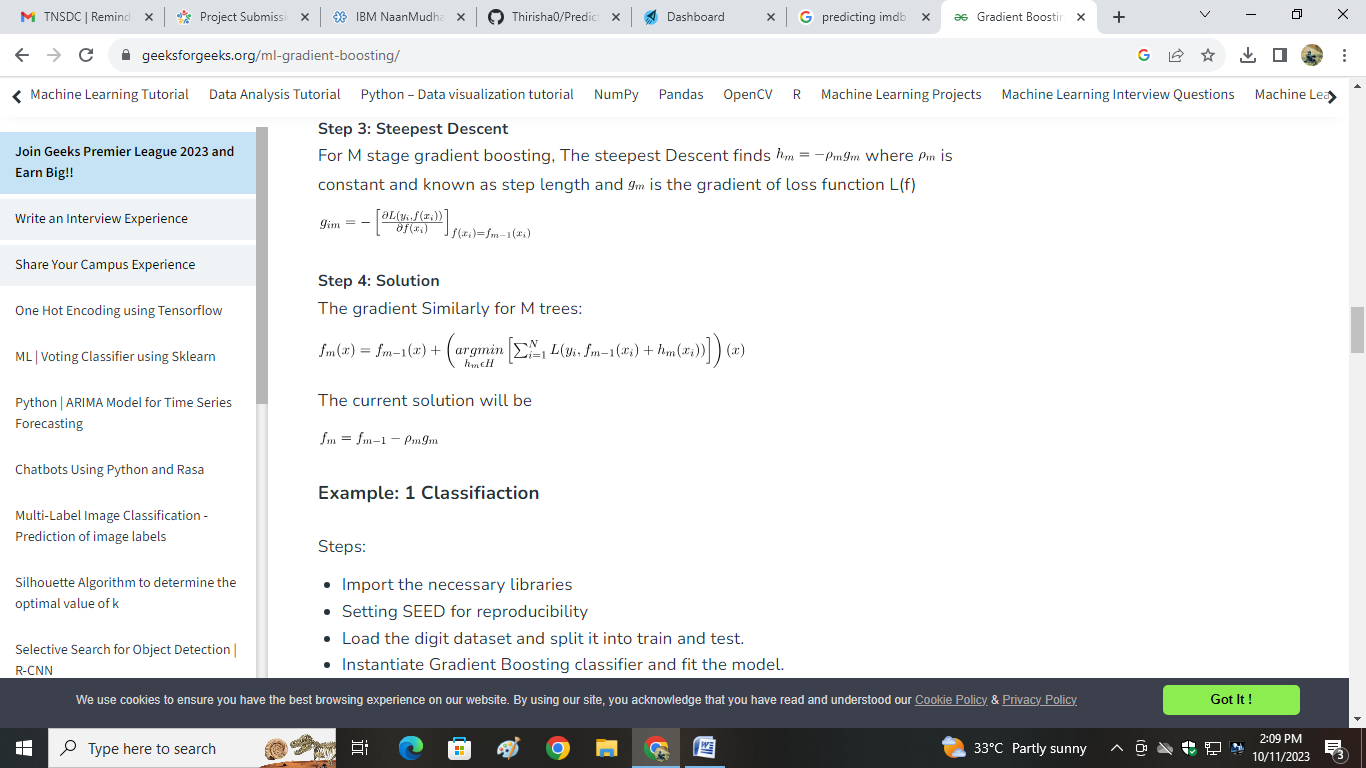
For M stage gradient boosting, The steepest Descent finds where is constant and known as step length and  is the gradient of loss function L(f)

**Step 4: Solution**

The gradient Similarly for M trees:



The current solution will be



**Program:**

# Import models and utility functions

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

from sklearn.datasets import load\_digits

# Setting SEED for reproducibility

SEED = 23

# Importing the dataset

X, y = load\_digits(return\_X\_y=True)

# Splitting dataset

train\_X, test\_X, train\_y, test\_y = train\_test\_split(X, y,

test\_size = 0.25,

random\_state = SEED)

# Instantiate Gradient Boosting Regressor

gbc = GradientBoostingClassifier(n\_estimators=300,

learning\_rate=0.05,

random\_state=100,

max\_features=5 )

# Fit to training set

gbc.fit(train\_X, train\_y)

# Predict on test set

pred\_y = gbc.predict(test\_X)

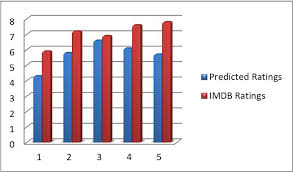
# accuracy

acc = accuracy\_score(test\_y, pred\_y)

print("Gradient Boosting Classifier accuracy is : {:.2f}".format(acc))

**Output:**

Gradient Boosting Classifier accuracy is : 0.98



**Data Loading:**

# Pandas - DataFrame - Loading the dataset from various data sources

A dataset can be loaded from various data sources using relevant Pandas constructs (functions) as mentioned below:

* CSV file - read\_csv() function
* JSON file - read\_json() function
* Excel file - read\_excel() function
* Database table - read\_sql() function

All the above functions return a dataframe object and most of these functions have a parameter called 'chunksize'.

e.g. to load a JSON data file (myfile.json) you can use the below code

my\_df = pd.read\_json("myfile.json")

Here, my\_df is a pandas dataframe object.

chunksize - It is the number of rows(records) of the dataset (csv, excel, json, table, etc.) which you want to be returned in each chunk.

When you use this parameter - chunksize, these functions (read\_csv(), read\_sql(), etc.) return you an iterator which enable you to traverse through these chunks of data, where each chunk is of size as specified by chunksize parameter.

This 'chunksize' parameter is very useful when you are dealing with (loading) a large dataset and you have very limited memory (RAM) available on your machine. If 'chunksize' parameter is specified, only a chunk of data will be read into the dataframe at a time. Hence, if your specified chunksize is within your memory (RAM) limits, you can easily load large datasets using these constructs/functions of Pandas.

**Instructions:**

Note: These instructions assume that you have completed the chapter on Numpy and have the necessary file housing\_short.csv in the correct directory. In case you don't have please go to the chapter of Numpy and complete the section on loading text file data first.

Please follow the below steps:

Please import pandas as pd

**Loading dataset from a CSV file**

(1) Please load the data from /cxldata/datasets/project/housing\_short.csv file by passing it to the read\_csv() function of Pandas library and store the returned dataframe in a variable called 'mydf'

**Data Preprocessing:**

Data preprocessing in Machine Learning is a crucial step that helps enhance the quality of data to promote the extraction of meaningful insights from the data. Data preprocessing in Machine Learning refers to the technique of preparing (cleaning and organizing) the raw data to make it suitable for a building and training Machine Learning models. In simple words, data preprocessing in Machine Learning is a [data mining technique](https://www.upgrad.com/blog/most-common-examples-of-data-mining/) that transforms raw data into an understandable and readable format.

Data preprocessing steps are a part of the data analysis and mining process responsible for converting raw data into a format understandable by the ML algorithms.

Text, photos, video, and other types of unprocessed, real-world data are disorganized. It may not only be inaccurate and inconsistent, but it is frequently lacking and doesn’t have a regular, consistent design. Machines prefer to process neat and orderly information; they read data as binary – 1s and 0s.

So, it is simple to calculate structured data like whole numbers and percentages. But before analysis, unstructured data, such as text and photos, must be prepped and formatted with the help of data preprocessing in Machine Learning.

Now that you know what is data preprocessing in machine learning, explore the major tasks in data preprocessing.

## Data Preprocessing Steps In Machine Learning: Major Tasks Involved:

Data cleaning, Data transformation, Data reduction, and Data integration are the major steps in data preprocessing.

### Data Cleaning

Data cleaning, one of the major preprocessing steps in machine learning, locates and fixes errors or discrepancies in the data. From duplicates and outliers to missing numbers, it fixes them all. Methods like transformation, removal, and imputation help ML professionals perform data cleaning seamlessly.

### Data Integration

Data integration is among the major responsibilities of data preprocessing in machine learning. This process integrates (merges) information extracted from multiple sources to outline and create a single dataset. The fact that you need to handle data in multiple forms, formats, and semantics makes data integration a challenging task for many ML developers.

### Data Transformation

ML programmers must pay close attention to data transformation when it comes to data preprocessing steps. This process entails putting the data in a format that will allow for analysis. Normalization, standardization, and discretisation are common data transformation procedures. While standardization transforms data to have a zero mean and unit variance, normalization scales data to a common range. Continuous data is discretized into discrete categories using this technique.

### Data Reduction

Data reduction is the process of lowering the dataset’s size while maintaining crucial information. Through the use of feature selection and feature extraction algorithms, data reduction can be accomplished. While feature extraction entails translating the data into a lower-dimensional space while keeping the crucial information, feature selection requires choosing a subset of pertinent characteristics from the dataset.

## Why Data Preprocessing in Machine Learning?

When it comes to creating a Machine Learning model, data preprocessing is the first step marking the initiation of the process. Typically, real-world data is incomplete, inconsistent, inaccurate (contains errors or outliers), and often lacks specific attribute values/trends. This is where data preprocessing enters the scenario – it helps to clean, format, and organize the raw data, thereby making it ready-to-go for Machine Learning models. Let’s explore various steps of data preprocessing in machine learning.

## Steps in Data Preprocessing in Machine Learning:

 There are seven significant steps in data preprocessing in Machine Learning:

### ****Acquire the dataset****

Acquiring the dataset is the first step in data preprocessing in machine learning. To build and develop Machine Learning models, you must first acquire the relevant dataset. This dataset will be comprised of data gathered from multiple and disparate sources which are then combined in a proper format to form a dataset. Dataset formats differ according to use cases. For instance, a business dataset will be entirely different from a medical dataset. While a business dataset will contain relevant industry and business data, a medical dataset will include healthcare-related data.

### Import all the crucial libraries

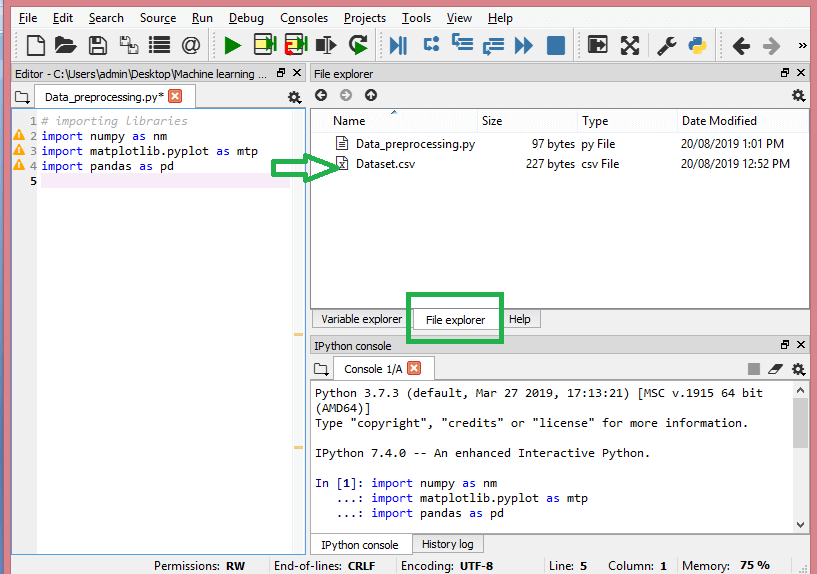
Since Python is the most extensively used and also the most preferred library by Data Scientists around the world, we’ll show you how to import Python libraries for data preprocessing in Machine Learning. Read more about [Python libraries for Data Science here.](https://www.upgrad.com/blog/python-libraries-for-data-science/) The predefined Python libraries can perform specific data preprocessing jobs. Importing all the crucial libraries is the second step in data preprocessing in machine learning. The three core Python libraries used for this data preprocessing in Machine Learning are:

* **NumPy** – NumPy is the fundamental package for scientific calculation in Python. Hence, it is used for inserting any type of mathematical operation in the code. Using NumPy, you can also add large multidimensional arrays and matrices in your code.
* **Pandas** – Pandas is an excellent open-source Python library for data manipulation and analysis. It is extensively used for importing and managing the datasets. It packs in high-performance, easy-to-use data structures and data analysis tools for Python.
* **Matplotlib** – Matplotlib is a Python 2D plotting library that is used to plot any type of charts in Python. It can deliver publication-quality figures in numerous hard copy formats and interactive environments across platforms (IPython shells, Jupyter notebook, web application servers, etc.).

### Import the dataset

In this step, you need to import the dataset/s that you have gathered for the ML project at hand. Importing the dataset is one of the important steps in data preprocessing in machine learning. However, before you can import the dataset/s, you must set the current directory as the working directory. You can set the working directory in Spyder IDE in three simple steps:

1. Save your Python file in the directory containing the dataset.
2. Go to File Explorer option in Spyder IDE and choose the required directory.
3. Now, click on the F5 button or Run option to execute the file.



**How to extract the independent variables?**

To extract the independent variables, you can use “iloc[ ]” function of the Pandas library. This function can extract selected rows and columns from the dataset.

x= data\_set.iloc[:,:-1].values

In the line of code above, the first colon(:) considers all the rows and the second colon(:) considers all the columns. The code contains “:-1” since you have to leave out the last column containing the dependent variable. By executing this code, you will obtain the matrix of features, like this –

[[‘India’ 38.0 68000.0]

 [‘France’ 43.0 45000.0]

 [‘Germany’ 30.0 54000.0]

 [‘France’ 48.0 65000.0]

 [‘Germany’ 40.0 nan]

 [‘India’ 35.0 58000.0]

 [‘Germany’ nan 53000.0]

 [‘France’ 49.0 79000.0]

 [‘India’ 50.0 88000.0]

 [‘France’ 37.0 77000.0]]

**How to extract the dependent variable?**

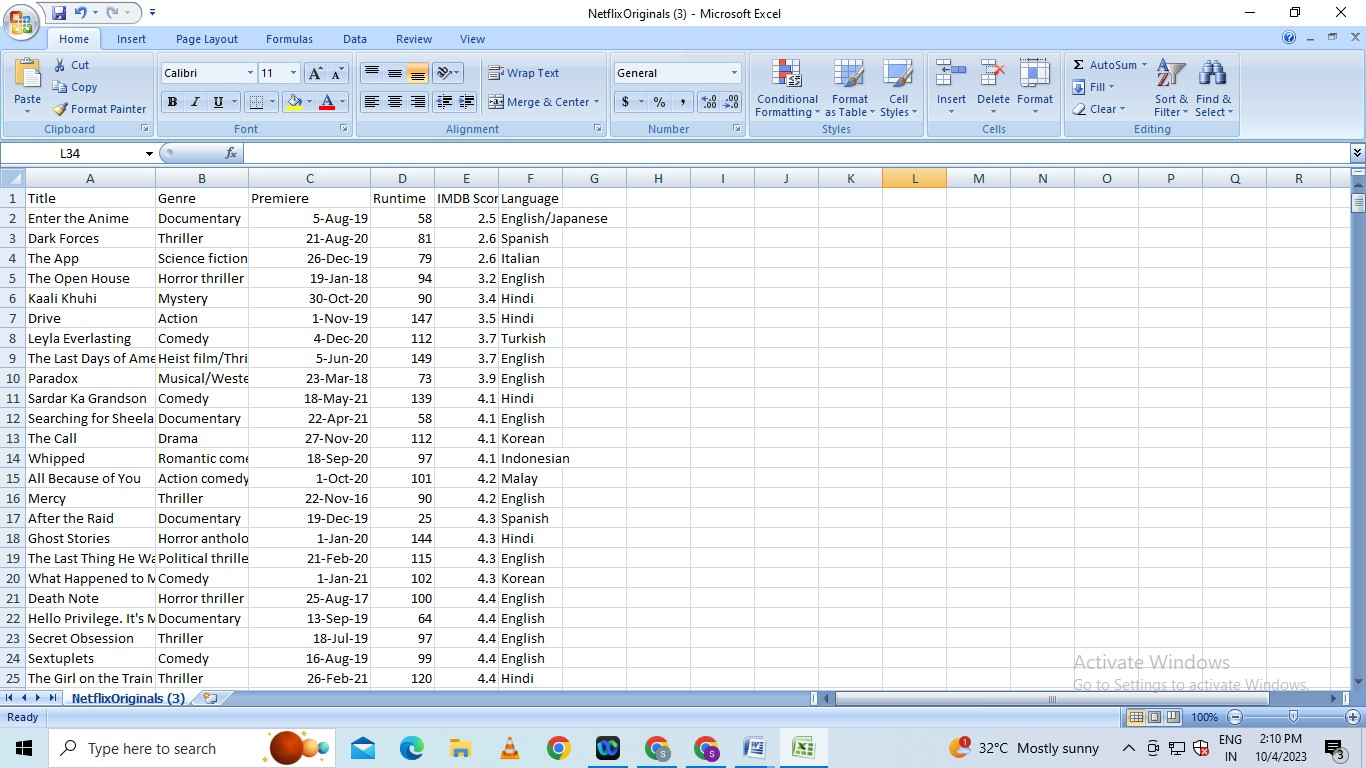
You can use the “iloc[ ]” function to extract the dependent variable as well. Here’s how you write it:

y= data\_set.iloc[:,3].values

This line of code considers all the rows with the last column only. By executing the above code, you will get the array of dependent variables, like so –

array([‘No’, ‘Yes’, ‘No’, ‘No’, ‘Yes’, ‘Yes’, ‘No’, ‘Yes’, ‘No’, ‘Yes’],

      dtype=object)

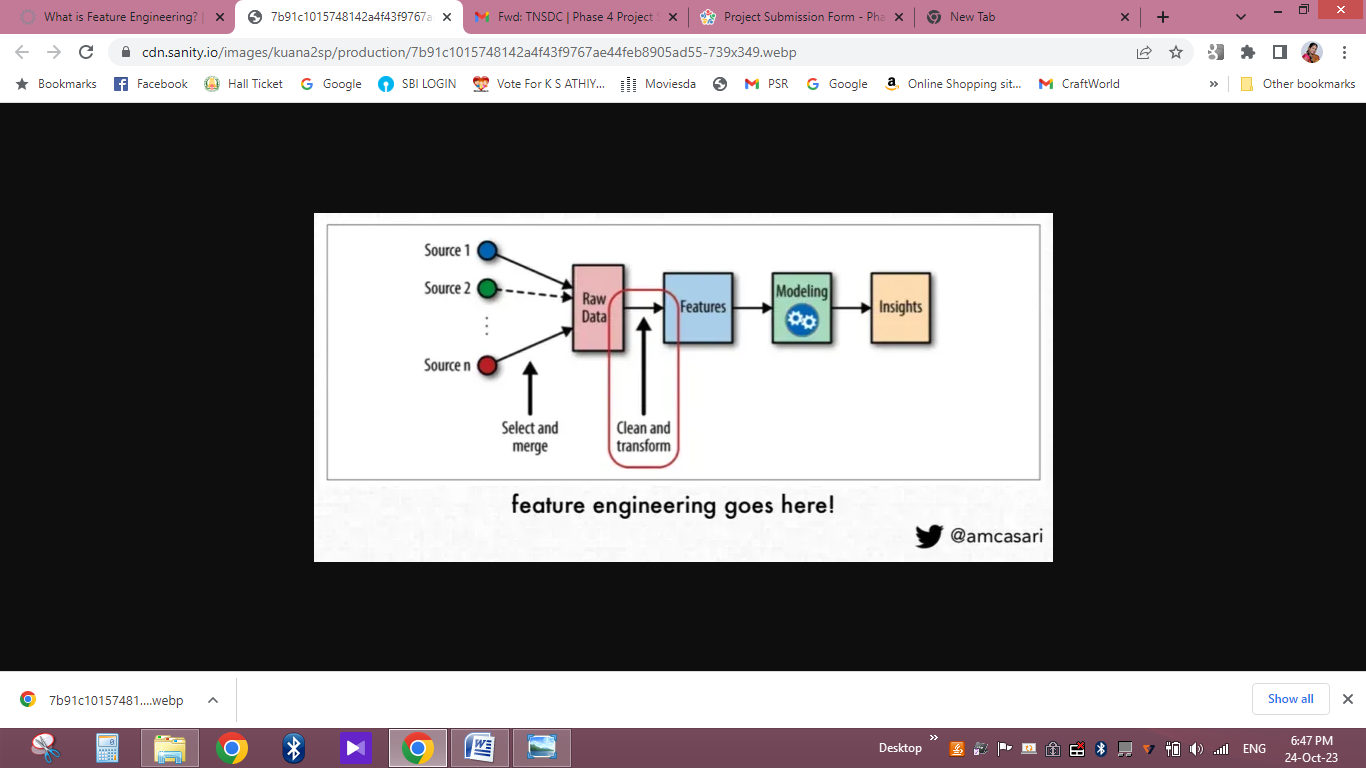


**What is Feature Engineering?**

Feature engineering refers to manipulation — addition, deletion, combination, mutation — of your data set to improve machine learning model training, leading to better performance and greater accuracy. Effective feature engineering is based on sound knowledge of the business problem and the available data sources.

Creating new features gives you a deeper understanding of your data and results in more valuable insights. When done correctly, feature engineering is one of the most valuable techniques of [data science](https://domino.ai/data-science-dictionary/data-science/), but it is also one of the most challenging. A common example of feature engineering is when your doctor uses your body mass index (BMI). BMI is calculated from both body weight and height, and serves as a surrogate for a characteristic that is very hard to accurately measure: the proportion of lean body mass.

**Feature Engineering in ML Lifecycle:**



Some common types of feature engineering include:

* **Scaling and normalization** means adjusting the range and center of data to ease learning and improve the interpretation of the results.
* **Filling missing values** implies filling in null values based on expert knowledge, heuristics, or by some [machine learning](https://domino.ai/data-science-dictionary/machine-learning/) techniques. Real-world datasets can be missing values due to the difficulty of collecting complete datasets and because of errors in the data collection process.
* **Feature selection** means removing features because they are unimportant, redundant, or outright counterproductive to learning. Sometimes you simply have too many features and need fewer.
* **Feature coding** involves choosing a set of symbolic values to represent different categories. Concepts can be captured with a single column that comprises multiple values, or they can be captured with multiple columns, each of which represents a single value and has a true or false in each field. For example, feature coding can indicate whether a particular row of data was collected on a holiday. This is a form of feature construction.
* **Feature construction** creates a new feature(s) from one or more other features. For example, using the date you can add a feature that indicates the day of the week. With this added insight, the algorithm could discover that certain outcomes are more likely on a Monday or a weekend.
* **Feature extraction** means moving from low-level features that are unsuitable for learning — practically speaking, you get poor testing results — to higher-level features that are useful for learning. Often feature extraction is valuable when you have specific data formats — like images or text — that have to be converted to a tabular row-column, example-feature format.

**What is Model Training?**

Model training is at the heart of the data science development lifecycle where the data science team works to fit the best weights and biases to an algorithm to minimize the loss function over prediction range. Loss functions define how to optimize the ML algorithms. A data science team may use different types of loss functions depending on the project objectives, the type of data used and the type of algorithm.

When a supervised learning technique is used, model training creates a mathematical representation of the relationship between the data features and a target label. In unsupervised learning, it creates a mathematical representation among the data features themselves.

**Importance of Model Training:**

Model training is the primary step in machine learning, resulting in a working model that can then be validated, tested and deployed. The model’s performance during training will eventually determine how well it will work when it is eventually put into an application for the end-users.

Both the quality of the training data and the choice of the algorithm are central to the model training phase. In most cases, training data is split into two sets for training and then validation and testing.

The selection of the algorithm is primarily determined by the end-use case. However, there are always additional factors that need to be considered, such as algorithm-model complexity, performance, interpretability, computer resource requirements, and speed. Balancing out these various requirements can make selecting algorithms an involved and complicated process.

**How To Train a Machine Learning Model?**

Training a model requires a systematic, repeatable process that maximizes your utilization of your available training data and the time of your data science team. Before you begin the training phase, you need to first determine your problem statement, access your data set and clean the data to be presented to the model.

In addition to this, you need to determine which algorithms you will use and what parameters (hyperparameters) they will run with. With all of this done, you can split your dataset into a training set and a testing set, then prepare your model algorithms for training.

**Split the Dataset**

Your initial training data is a limited resource that needs to be allocated carefully. Some of it can be used to train your model, and some of it can be used to test your model – but you can’t use the same data for each step. You can’t properly test a model unless you have given it a new data set that it hasn’t encountered before. Splitting the training data into two or more sets allows you to train and then validate the model using a single source of data. This allows you to see if the model is overfit, meaning that it performs well with the training data but poorly with the test data.

A common way of splitting the training data is to use cross-validation. In [10-fold cross-validation](https://domino.ai/blog/guide-to-building-models-with-cross-validation), for example, the data is split into ten sets, allowing you to train and test the data ten times.

**Select Algorithms to Test**

In machine learning, there are thousands of algorithms to choose from, and there is no sure way to determine which will be the best for any specific model. In most cases, you will likely try dozens, if not hundreds, of algorithms in order to find the one that results in an accurate working model.

**Tune the Hyperparameters**

Hyperparameters are the high-level attributes set by the data science team before the model is assembled and trained. While many attributes can be learned from the training data, they cannot learn their own hyperparameters.

As an example, if you are using a [regression algorithm](http://www.sthda.com/english/articles/37-model-selection-essentials-in-r/153-penalized-regression-essentials-ridge-lasso-elastic-net/), the model can determine the regression coefficients itself by analyzing the data. However, it cannot dictate the strength of the penalty it should use to regularize an overabundance of variables. As another example, a model using the random forest technique can determine where decision trees will be split, but the number of trees to be used needs to be tuned beforehand.

**Fit and Tune Models**

Now that the data is prepared and the model’s hyperparameters have been determined, it’s time to start training the models. The process is essentially to loop through the different algorithms using each set of hyperparameter values you’ve decided to explore.

Next, select another set of hyperparameter values you want to try for the same algorithm, cross-validate it again and calculate the new score. Once you have tried each hyperparameter value, you can repeat these same steps for additional algorithms.

Think of these trials as track and field heats. Each algorithm has demonstrated what it can do with the different hyperparameter values. Now you can select the best version from each algorithm and send them on to the final competition.

**Choose the Best Model**

Now it’s time to test the best versions of each algorithm to determine which gives you the best. Once the testing is done, you can compare their performance to determine which are the better models. The overall winner should have performed well (if not the best) in training as well as in testing. It should also perform well on your other performance metrics (like speed and [empirical loss](https://developers.google.com/machine-learning/crash-course/descending-into-ml/training-and-loss)), and – ultimately – it should adequately solve or answer the question posed in your problem statement.

**What is Model Evaluation?**

Model evaluation is the process of using different evaluation metrics to understand a machine learning model’s performance, as well as its strengths and weaknesses. Model evaluation is important to assess the efficacy of a model during initial research phases, and it also plays a role in model monitoring.

To understand if your model(s) is working well with new data, you can leverage a number of evaluation metrics.

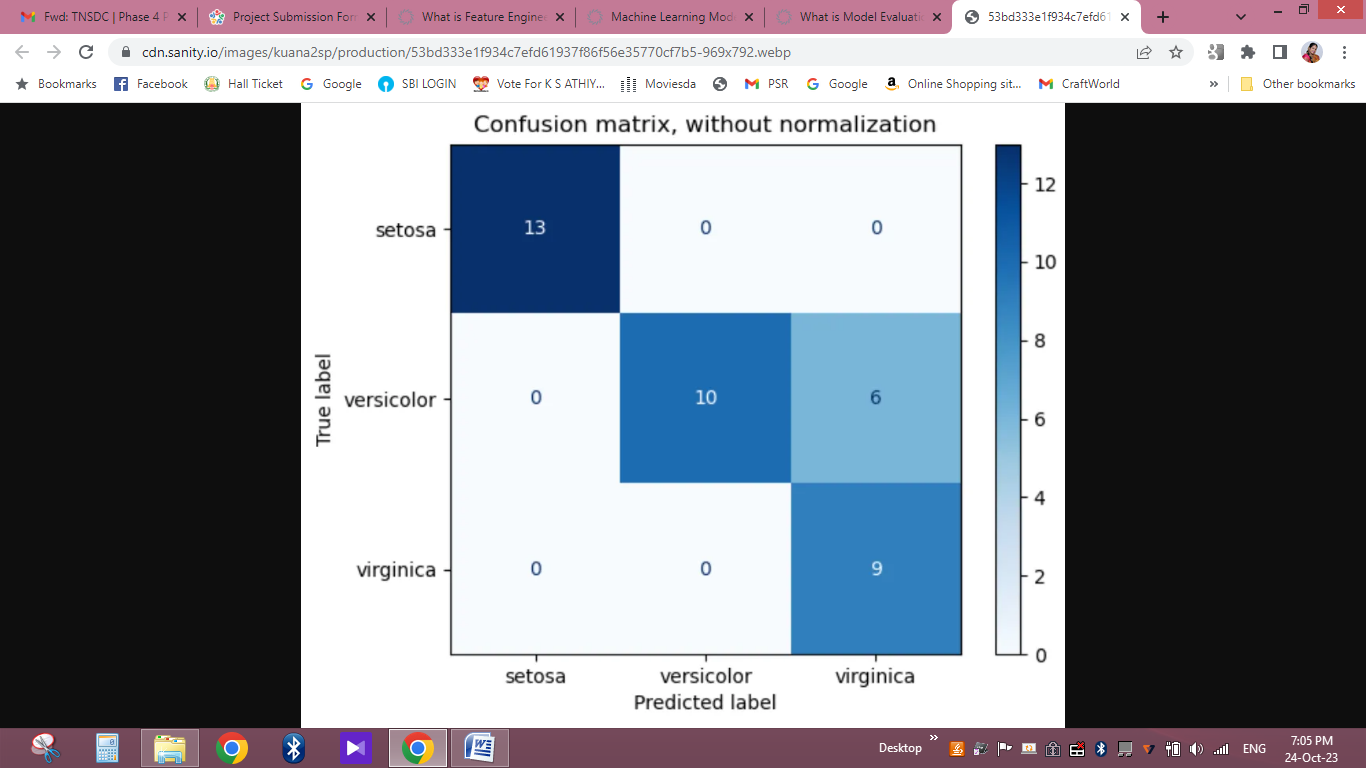
**Classification:**

The most popular metrics for measuring classification performance include accuracy, precision, confusion matrix, log-loss, and AUC (area under the ROC curve).

* **Accuracy** measures how often the classifier makes the correct predictions, as it is the ratio between the number of correct predictions and the total number of predictions.
* **Precision** measures the proportion of predicted Positives that are truly Positive. Precision is a good choice of evaluation metrics when you want to be very sure of your prediction. For example, if you are building a system to predict whether to decrease the credit limit on a particular account, you want to be very sure about the prediction or it may result in customer dissatisfaction.
* The **confusion matrix** (or confusion table) shows a more detailed breakdown of correct and incorrect classifications for each class. Using a confusion matrix is useful when you want to understand the distinction between classes, particularly when the cost of misclassification might differ for the two classes, or you have a lot more test data on one class than the other. For example, the consequences of making a false positive or false negative in a cancer diagnosis are very different.
* **Log-loss** (logarithmic loss) can be used if the raw output of the classifier is a numeric probability instead of a class label. The probability can be understood as a gauge of confidence, as it is a measurement of accuracy.
* **AUC** (Area Under the ROC Curve) is a performance measurement for classification problems at various thresholds settings. It tells how much a model is capable of distinguishing between classes. The higher the AUC, better the model is at predicting when a 0 is actually a 0 and a 1 is actually a 1. Similarly, the higher the AUC, the better the model is at distinguishing between patients with a disease and with no disease.

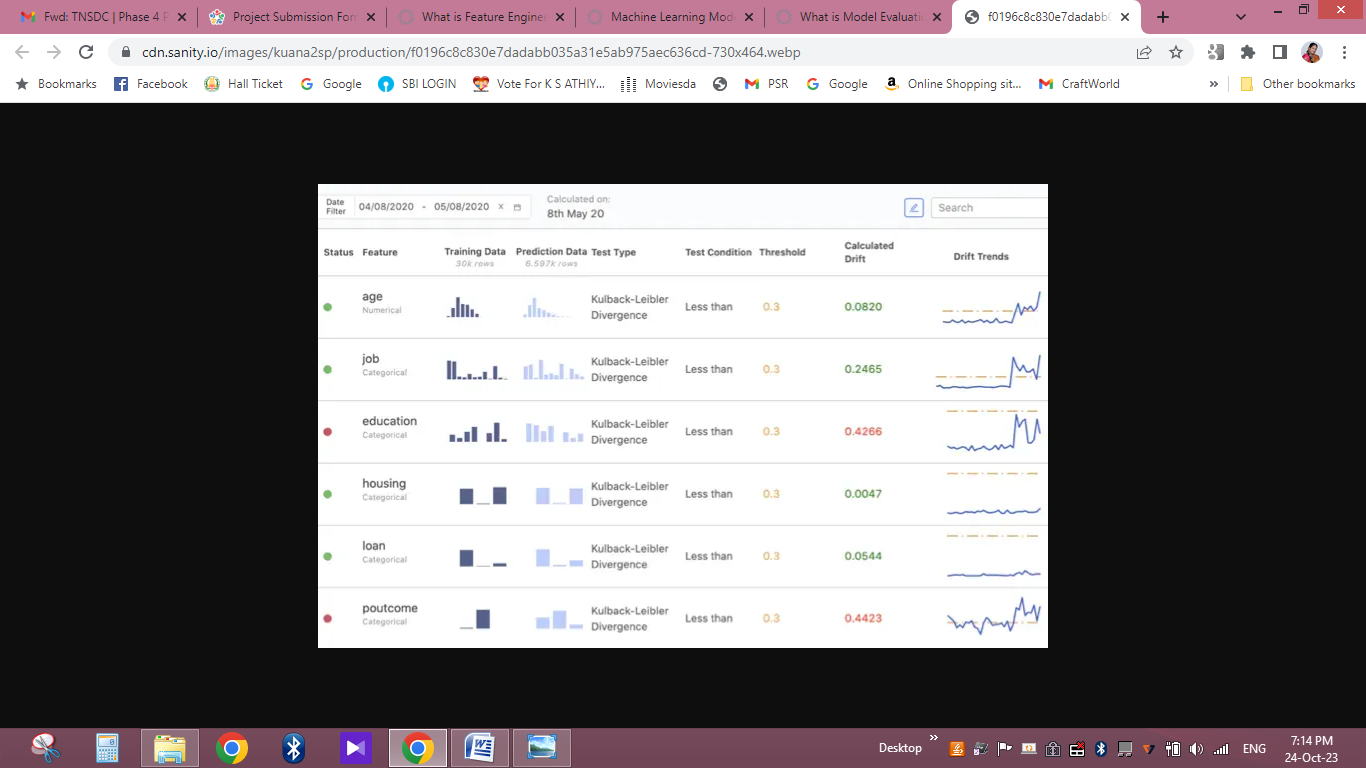
Other popular metrics exist for regression models, like R Square, Adjusted R Square, MSE (Mean Squared Error), RMSE (Root Mean Squared Error), and MAE (Mean Absolute Error).

**Example of Confusion Matrix on Iris Flower Dataset:**



**Domino Model Monitor:**

Machine Learning Operations teams often monitor multiple models at once by checking model predictions, checking (input) data drift, and checking concept drift. Model monitoring  tools, like Domino Model Monitor,



**Conclusion:**

IMDB ratings are a popular way to gauge the quality of a movie. However, they can be subjective and vary from person to person. In order to create a more objective measure of movie quality, researchers have developed a number of methods for predicting IMDB scores.

One common approach is to use machine learning algorithms to train a model on a dataset of movies and their IMDB ratings. The model can then be used to predict the IMDB rating of a new movie. This approach has been shown to be effective in predicting IMDB ratings, but it can be difficult to train a model that is able to generalize to new movies.

Another approach is to use a rule-based system to predict IMDB ratings. This system would use a set of rules to assign a rating to a movie based on its genre, director, cast, and other factors. This approach is less accurate than machine learning, but it is easier to implement and can be used to predict the ratings of movies that are not in the training dataset.

Overall, there are a number of different methods for predicting IMDB scores. The best approach will depend on the specific application. For example, if you need to predict the ratings of a large number of movies, then machine learning is a good option. However, if you need to predict the ratings of a small number of movies or movies that are not in the training dataset, then a rule-based system may be a better choice

|  |
| --- |
|  |