**PPT**

I chose to work on the IMDB score predictions project for several reasons. Firstly, I was interested in the challenge of using machine learning algorithms to predict the scores of movies and TV shows based on a variety of factors, such as genre, cast, and production budget. I felt that this project would allow me to develop my skills in data analysis and modelling, and to gain a deeper understanding of how machine learning algorithms can be used to solve real-world problems.

**Abstract**

* IMDb is a website owned by Amazon.com that provides information about movies to its users. It's a valuable resource for anyone interested in movies, whether it's for personal interest or professional research.
* This line explains that a commercially successful movie not only entertains audiences but also generates significant profits for film companies. It also states that while factors like having a good director and experienced actors are important for creating good movies, they cannot guarantee a high IMDb score, which is a measure of critical acclaim and audience appreciation.
* which is to predict the rating of a movie by utilizing Machine Learning models that have been properly trained. The goal is to evaluate the accuracy and reliability of the predictions made by these models. This line can be used in an interview to illustrate how Machine Learning can be applied to predict the success or performance of a movie and how it can be useful for film companies and movie industry professionals.

**Dataset Description**

This line explains the focus of a project that aims to predict IMDB scores for movies using a dataset obtained from the Kaggle website. The response variable in the project is the IMDB score, and the goal is to analyze the other variables in the movie\_metadata to make predictions. The dataset includes information about 5043 movies spanning 100 years and 66 countries, with thousands of actors/actresses and 2399 unique director names. This line can be used in an interview to highlight the scope and scale of a data analysis project and how it can involve exploring large and complex datasets.

**Data mining**

This line defines data mining as a process of extracting information from large datasets to identify patterns, trends, and useful data that can enable businesses to make data-driven decisions. Data mining involves using various statistical and computational techniques to analyze data and extract insights that can inform decision-making. This line can be used in an interview to provide a concise definition of data mining and its relevance to businesses in today's data-driven economy.

**Data cleaning**

Data cleaning refers to the process of identifying and correcting errors, inconsistencies, and inaccuracies in a dataset. This involves detecting and removing duplicate entries, correcting missing or incorrect data, standardizing data formats, and resolving inconsistencies between different data sources.

**Data Exploration**

Data Exploration is the process of analyzing and summarizing data to gain insights and understanding about its characteristics, relationships, and patterns. It involves using various techniques and tools to explore the data visually and statistically, such as descriptive statistics, data visualization, and hypothesis testing. The goal of data exploration is to gain a deeper understanding of the data and identify potential areas for further analysis.

**Feature Engineering**

Feature engineering is the process of selecting, creating, and transforming variables, or features, in a dataset to improve the performance of machine learning models. It involves identifying the most relevant and informative features that can help the model make accurate predictions, and transforming them into a format that is suitable for analysis. Feature engineering can involve a range of techniques, including encoding categorical variables, scaling numerical variables, creating interaction terms, and selecting subsets of features through feature selection methods. The goal of feature engineering is to enhance the predictive power of machine learning models by providing them with more informative and relevant inputs.

**Predictive Modelling**

Predictive modeling is the process of using statistical or machine learning algorithms to make predictions about future events or behaviors based on historical data. It involves building a model using a training dataset that captures the relationships between the predictor variables (features) and the target variable (outcome). The model is then used to make predictions on a new dataset that it has not seen before, known as the test dataset. Predictive modeling can be used in various domains, such as finance, healthcare, marketing, and social sciences, to make predictions about customer behavior, disease diagnosis, stock prices, and more. The goal of predictive modeling is to improve decision-making by providing accurate and reliable predictions based on historical data.

**JUPYTER**

Since the target variable "imdb\_score" is a continuous variable, we would use regression type algorithms to build a predictive model. Regression models are used to predict a continuous output variable based on one or more input variables or features. There are several types of regression algorithms, such as **linear regression, decision tree regression, random forest** regression, and support vector regression, that can be used depending on the specific requirements of the problem and the characteristics of the dataset. The ultimate goal of using regression models in this context is to accurately predict the imdb\_score for new data points based on the relationships between the input features and the target variable observed in the training dataset.

1. ***Libraries***

* NumPy: For numerical operations and working with arrays and matrices
* Pandas: For data manipulation and analysis
* Matplotlib: For data visualization
* Seaborn: For advanced data visualization
* Scikit-learn: For building machine learning models

**Load data**

1. **pd.read\_csv** is a function in the Pandas library of Python that is used to read a CSV (Comma Separated Values) file into a DataFrame object
2. **pd.set\_option** function to change the maximum number of rows and columns to display in a Pandas DataFrame:
3. **df.head()** is a method of a Pandas DataFrame object that is used to display the first n rows of the DataFrame
4. **df.shape** is an attribute of a Pandas DataFrame object that returns a tuple representing the dimensions of the DataFrame: the number of rows and columns, respectively.
5. **df.columns** is a property that returns the column labels of a DataFrame object as a pandas.Index object.

**Missing values**

When null values are caused by a technical issue: In some cases, null values in a dataset can be caused by a technical issue, such as data collection or data transfer errors. In these cases, it may be more appropriate to fix the technical issue and re-collect the data or transfer the data again, rather than removing the null values.

1. **df.isna().sum().sort\_values(ascending=False)** is a code snippet that returns the number of missing (NaN) values for each column in a DataFrame object in descending order.
2. **df.dropna(subset=['gross', 'budget'], inplace=True)** is a method that drops the rows containing missing (NaN) values in specified columns of a DataFrame object.

what is the reasons for drop all null values from rows wise in our dataset?

Improving accuracy: Null values in a dataset can cause inaccurate results if they are not handled properly. By dropping null values from rows, the remaining data can be more accurate and reliable, which can lead to better decision-making.

1. **df.isna().sum().sort\_values(ascending=False)** is a code snippet that returns the number of missing (NaN) values for each column in a DataFrame object in descending order.
2. **df.dropna(subset=['gross', 'budget'], inplace=True)** is a method that drops the rows containing missing (NaN) values in specified columns of a DataFrame object.

what is the reasons for drop all null values from rows wise in our dataset?

Improving accuracy: Null values in a dataset can cause inaccurate results if they are not handled properly. By dropping null values from rows, the remaining data can be more accurate and reliable, which can lead to better decision-making.

1. **df.shape** is an attribute of a Pandas DataFrame object that returns a tuple representing the dimensions of the DataFrame: the number of rows and columns, respectively.
2. **df.isna().sum()** This tells us that column 'A' has 10 missing values, column 'B' has 5 missing values, and column 'C' has 0 missing values.

**Remove duplicates**

1. **df.duplicated().sum()** is that computes the number of duplicate rows in a Pandas DataFrame

*(Duplicate rows can be a common issue in real-world datasets, and they can cause problems in data analysis or machine learning modeling if they are not handled properly. By identifying and removing duplicate rows, we can ensure that our analysis or modeling is based on unique and valid data.)*

1. **df.drop\_duplicates(keep='first', inplace=True)** that drops duplicate rows from a Pandas DataFrame

*(which removes rows that are duplicates of previous rows (i.e., identical to a previous row).By removing duplicate rows, we can ensure that our analysis or modeling is based on unique and valid data.)*

1. **df.shape** is an attribute of a Pandas DataFrame object that returns a tuple representing the dimensions of the DataFrame: the number of rows and columns, respectively.
2. **df.info()** is a method in Pandas that provides a summary of the DataFrame, including the index type, column names, non-null values, and data types for each column.
3. **df.describe().T** is a pandas DataFrame method that provides a summary of the statistical properties of a DataFrame. The **describe()** method calculates various descriptive statistics such as count, mean, standard deviation, minimum, maximum, and percentiles for each numerical column in the DataFrame. The **T** method transposes the output so that the columns become rows and the rows become columns
4. **df.columns** is a property that returns the column labels of a DataFrame object as a pandas.Index object.
5. **df.drop\_duplicates(keep='first', inplace=True)** that drops duplicate rows from a Pandas DataFrame

*(which removes rows that are duplicates of previous rows (i.e., identical to a previous row).By removing duplicate rows, we can ensure that our analysis or modeling is based on unique and valid data.)*

1. **df.drop\_duplicates(keep='first', inplace=True)** that drops duplicate rows from a Pandas DataFrame

*(which removes rows that are duplicates of previous rows (i.e., identical to a previous row).By removing duplicate rows, we can ensure that our analysis or modeling is based on unique and valid data.)*

1. **df.color.value\_counts()** is a pandas method that returns a count of unique values in the **color** column. The resulting output is a pandas Series that shows the count of each unique value in descending order. In this case, there are three occurrences of "blue", one occurrence of "green", and one occurrence of "red". This information can be useful for understanding the distribution of values in a categorical variable.
2. **df.language.value\_counts()** would return a count of unique language values in the **language** column. The resulting output is a pandas Series that shows the count of each unique value in descending order. In this case, there is only one unique language value, "English", and it occurs in all 5 rows of the DataFrame. This information may be useful for understanding the distribution of languages in the movie data and identifying potential biases or limitations in the dataset.
3. **df.drop\_duplicates(keep='first', inplace=True)** that drops duplicate rows from a Pandas DataFrame

*(which removes rows that are duplicates of previous rows (i.e., identical to a previous row).By removing duplicate rows, we can ensure that our analysis or modeling is based on unique and valid data.)*

1. **df.country.value\_counts()** would return a count of unique country values in the **country** column. The resulting output is a pandas Series that shows the count of each unique value in descending order. In this case, there is only one unique country value, "USA", and it occurs in all 5 rows of the DataFrame. This information may be useful for understanding the distribution of countries in the movie data and identifying potential biases or limitations in the dataset.
2. **Filter**

**grec** DataFrame to only include rows where the 'country' column is equal to 'India', and stores the result in a new DataFrame called **result**. The resulting DataFrame will contain the movie titles and country information for only those movies that were produced in India.

1. **Def** In this example, the resulting DataFrame has three unique country values: 'USA', 'UK', and 'Others'. The 'USA' and 'UK' values have been retained as distinct categories, while all other countries have been combined into a single 'Others' category. The **value\_counts()** method is used to count the number of movies in each country category.
2. **df['genres'].head()** returns the first five values of the 'genres' column in the DataFrame **df**. The 'genres' column contains a string of comma-separated genre labels for each movie. This output shows the first five values of the 'genres' column in **df**. Each value is a string of comma-separated genre labels.
3. **genre\_df = df[['genres', 'imdb\_score']]** creates a new DataFrame called **genre\_df** that contains only the 'genres' and 'imdb\_score' columns from the original DataFrame **df**. The new DataFrame has the same number of rows as the original DataFrame, but only contains two columns. **genre\_df** DataFrame has the same number of rows as the original DataFrame **df**, but only contains two columns: 'genres' and 'imdb\_score'. The **head()** method is used to display the first five rows of the new DataFrame.
4. **genre\_df=genre\_df.assign(genres=genre\_df['genres'].str.split('|')).explode('genres')** splits the 'genres' column on '|' separator and creates a new row for each genre value along with its corresponding IMDb score value. This is achieved using the **assign()** method to create a new column called 'genres' which contains a list of genre labels (obtained by using the **str.split()** method), and then using the **explode()** method to create a new row for each element of the 'genres' list. The resulting DataFrame has multiple rows for each movie, with each row containing a single genre label and its corresponding IMDb score value.
5. **genre\_df['genres'].value\_counts()** returns a Series object that contains the count of each unique value in the 'genres' column of the 'genre\_df' DataFrame. This is achieved using the **value\_counts()** method, which counts the occurrence of each value in the Series object and returns a new Series object with the counts. As we can see from the output, Drama is the most common genre, followed by Crime and Comedy.
6. **genre\_df.genres.unique()** returns an array of unique genre labels in the 'genres' column of the 'genre\_df' DataFrame. This is achieved using the **unique()** method, which returns an array of unique values in the specified Series object. As we can see from the output, there are five unique genre labels in the 'genres' column: Crime, Drama, Comedy, Musical, and Action.
7. **plt.figure(figsize=(9,4)) genre = sns.barplot(x='genres', y='imdb\_score', data = genre\_df, palette='turbo') plt.xticks(rotation=65, horizontalalignment='right') plt.show()** creates a bar plot of IMDb scores for each genre in the 'genre\_df' DataFrame. This is achieved using the **sns.barplot()** function from the Seaborn library, which creates a bar plot with one categorical variable on the x-axis and one numerical variable on the y-axis. Overall, this code creates a bar plot showing the average IMDb scores for each genre in the 'genre\_df' DataFrame, allowing us to compare the IMDb scores across different genres.
8. **df.drop\_duplicates(keep='first', inplace=True)** that drops duplicate rows from a Pandas DataFrame

*(which removes rows that are duplicates of previous rows (i.e., identical to a previous row).By removing duplicate rows, we can ensure that our analysis or modeling is based on unique and valid data.)*

1. **df.aspect\_ratio.value\_counts()** returns the count of unique values in the 'aspect\_ratio' column of the 'df' DataFrame, sorted in descending order. The 'aspect\_ratio' column contains the aspect ratios of the movies in the dataset, which is the ratio of the width to the height of the movie screen.

For example, if this code returns a count of 2345 for the aspect ratio value of 2.35, it means that there are 2345 movies in the dataset with an aspect ratio of 2.35:1. This information can be useful for analyzing the distribution of aspect ratios in the dataset, and for identifying any trends or patterns in the aspect ratios of movies over time.

1. **IMDb score for movies with aspect ratios of 2.35 and 1.85**, we can filter the 'df' DataFrame based on these aspect ratios and then calculate the mean of the 'imdb\_score' column for each subset of data. This code first creates a new DataFrame 'df\_filtered' by filtering the 'df' DataFrame to only include movies with aspect ratio values of 2.35 or 1.85, using the **isin()** method. The resulting DataFrame will have only the rows that meet this condition.Next, the code calculates the mean IMDb score for each aspect ratio value, using the **groupby()** method to group the data by 'aspect\_ratio' and then calculating the mean of the 'imdb\_score' column for each group.Finally, the code prints the mean scores for each aspect ratio value. The resulting output will show the mean IMDb scores for movies with aspect ratios of 2.35 and 1.85, respectively.
2. **Df.drop()** method with the **inplace=True** argument will modify the **df** DataFrame to remove the 'aspect\_ratio' column. Here is the updated code: This will drop the 'aspect\_ratio' column from the DataFrame, allowing us to focus on other variables that may be more useful for predicting IMDb scores.
3. **df.describe().T** is a pandas DataFrame method that provides a summary of the statistical properties of a DataFrame. The **describe()** method calculates various descriptive statistics such as count, mean, standard deviation, minimum, maximum, and percentiles for each numerical column in the DataFrame. The **T** method transposes the output so that the columns become rows and the rows become columns
4. **Replacing 0's** with NaN as you have done in your code can be a good way to handle missing data in the 'director\_facebook\_likes' column. This will convert all the 0's in the column to NaN values, which can make it easier to identify and handle missing values in the data.
5. **df.describe().T** is a pandas DataFrame method that provides a summary of the statistical properties of a DataFrame. The **describe()** method calculates various descriptive statistics such as count, mean, standard deviation, minimum, maximum, and percentiles for each numerical column in the DataFrame. The **T** method transposes the output so that the columns become rows and the rows become columns
6. **df['content\_rating'].value\_counts()**This code will display the count of movies for each content rating in the 'content\_rating' column of your dataframe 'df'. This can be useful to understand the distribution of content ratings in your dataset and identify any patterns or outliers.
7. **df['content\_rating'] = df['content\_rating'].replace(['M', 'GP'], 'PG')**These lines of code use the replace() function to replace 'X' with 'NC-17' and 'Approved', 'NotRated', 'Passed', and 'Unrated' with 'R'. By grouping these ratings together, you can simplify your analysis and focus on the most common content ratings in your dataset.
8. **df['profit'] = df.apply(lambda x: func(x['gross'], x['budget']), axis=1)** This code defines a function called "func" that takes in two arguments - 'gross' and 'budget' - and returns the difference between them. The code then uses the apply() function to apply this function to each row of the 'df' dataframe, passing in the 'gross' and 'budget' values for each row and assigning the resulting value to a new column called 'profit'. This defines a function called "func" that takes in two arguments - 'gross' and 'budget' - and returns their difference. he resulting 'profit' column will contain the difference between the 'gross' and 'budget' values for each movie in the dataset. This can be useful for analyzing the profitability of each movie and identifying patterns and outliers in the data.
9. **df['profit']**This column might represent the difference between a movie's gross revenue and its budget, indicating how much profit the movie made. For example, if a movie had a budget of $100 million and grossed $200 million, then the profit for that movie would be $100 million. The values in the 'profit' column could be positive, negative, or zero depending on the success of the movie.

**Data visualization**

is the process of creating graphical representations of data to help communicate insights and patterns in the data. Visualization can be a powerful tool for understanding complex datasets, identifying trends and outliers, and communicating findings to others.

Here are some common types of data visualizations:

1. Bar chart: A bar chart displays categorical data as rectangular bars with heights proportional to the values they represent.
2. Line chart: A line chart displays data as a series of points connected by straight lines. This is useful for showing trends over time or other continuous variables.
3. Scatter plot: A scatter plot displays the relationship between two continuous variables as a set of points on a two-dimensional graph.
4. Heatmap: A heatmap displays data as a grid of colored squares, with colors indicating the value of each square. This is useful for visualizing patterns in large datasets.
5. Pie chart: A pie chart displays data as a circular graph, with slices representing different categories and their relative sizes indicating the proportion of each category.
6. Histogram: A histogram displays the distribution of a continuous variable by dividing it into bins and displaying the frequency of observations in each bin as a bar.

There are many other types of data visualizations, and the choice of visualization depends on the type of data and the research question being addressed. By creating clear and informative visualizations, you can help to better understand your data and communicate your findings to others.

1. **plt.figure(figsize=(8,4)) sns.histplot(df.title\_year, color='Green') plt.show()** This code creates a histogram of the distribution of movie release years in the 'df' dataframe using the seaborn library. The resulting histogram will show the distribution of movie release years in the 'df' dataframe. This can be useful for understanding trends and patterns in the data, such as whether there has been an increase or decrease in movie production over time.
2. **as more movies are after year 1980 # we'll consider only movies after 1980 df = df[df.title\_year>=1980]** his code filters the 'df' dataframe to include only movies released after 1980 he resulting 'df' dataframe will only contain movies released after 1980, which can be useful for analyzing more recent trends and patterns in the data. However, it's important to keep in mind that this filter may exclude important data from earlier years, so it should be used judiciously based on the research question being addressed.
3. **df.title\_year.value\_counts().sort\_index()** This code counts the number of movies released in each year in the 'df' dataframe and sorts the results by year in ascending order. The resulting output will show the number of movies released in each year in the 'df' dataframe, sorted by year in ascending order. This can be useful for understanding trends and patterns in the data over time, such as identifying years with particularly high or low numbers of movie releases.
4. **top 20 movies based on its Profit top\_movies\_profit = df[['movie\_title', 'profit', 'budget']].sort\_values('profit', ascending=False)[:20] top\_movies\_profit**

This code selects the top 20 movies from the 'df' dataframe based on their profit, and creates a new dataframe with columns for the movie title, profit, and budget.he resulting 'top\_movies\_profit' dataframe will show the top 20 movies in the 'df' dataframe based on their profit, sorted in descending order. This can be useful for understanding which movies were most financially successful, and for identifying common themes or trends among these successful movies.

1. This code creates a **scatter plot** of the top 20 movies in the 'df' dataframe based on their profit, with the profit values plotted on the x-axis and the budget values plotted on the y-axis. The movie titles are displayed as text labels on the plot. The resulting scatter plot will show the top 20 movies in the 'df' dataframe based on their profit, with the profit values plotted on the x-axis and the budget values plotted on the y-axis. The movie titles are displayed as text labels on the plot, and blue triangle markers indicate the (x,y) coordinates for each movie. This can be useful for understanding the relationship between movie budgets and profits, and for identifying movies that were particularly successful despite having relatively low budgets.
2. **df.director\_name.value\_counts().nlargest(20)** This code counts the number of movies in the 'df' dataframe that each director has directed, and then selects the top 20 directors with the most movies. It returns a Pandas series with the director names as the index and the number of movies as the values, sorted in descending order. The resulting Pandas series will show the top 20 directors in the 'df' dataframe based on the number of movies they have directed. This can be useful for understanding which directors have been most prolific in their careers, and for identifying common themes or trends in their work.
3. his code creates a new dataframe 'direc\_df' by selecting only the 'director\_name' and 'imdb\_score' columns from the 'df' dataframe. Then, it creates a new dataframe 'result' by selecting only the rows from 'direc\_df' where the 'imdb\_score' is greater than 6. The resulting 'result' dataframe will contain only movies with an imdb\_score greater than 6. This can be useful for identifying highly-rated movies from directors who have a history of producing quality work. It can also help identify trends or patterns in the types of movies that tend to have high imdb\_scores.
4. **Data Preprocessing**

**print(df['director\_name'].nunique()) # unique values of director print(df[['actor\_1\_name', 'actor\_2\_name', 'actor\_3\_name']].stack().nunique()) # unique values of actors**

This is useful for understanding how many unique actors are represented in the dataset, taking into account that a single movie may have multiple actors**.**

1. **df.drop\_duplicates(keep='first', inplace=True)** that drops duplicate rows from a Pandas DataFrame

*(which removes rows that are duplicates of previous rows (i.e., identical to a previous row).By removing duplicate rows, we can ensure that our analysis or modeling is based on unique and valid data.)*

1. **Remove Linear Dependent Variables**

**df.drop(['profit'], axis=1, inplace=True)**

Linear dependence occurs when two or more predictor variables are highly correlated with each other, which can result in unstable and unreliable regression models. However, removing a variable from the dataset solely based on its name ('profit') without considering its correlation with other variables may result in losing important information and affecting the accuracy of the predictive model. It is important to carefully analyze the dataset and identify the relevant variables for the prediction task before making any changes.

1. **Remove Highly Correlated Variables**

**# Generate a mask for the upper triangle #mask = np.triu(np.ones\_like(corr, dtype=bool)) plt.figure(figsize=(13,6)) sns.heatmap(df.corr(), annot=True) plt.show()**

The code provided generates a correlation heatmap using Seaborn's **heatmap()** function. The heatmap visualizes the pairwise correlations between all the variables in the dataframe **df** using a color scale.

The **mask** is created using **np.triu()** to obtain only the upper triangle of the matrix, excluding the diagonal.

**58. Finally, plt.show() displays the heatmap figure.**

Overall, this code is useful for visualizing the correlation matrix of the variables in **df** and identifying potential relationships between them. By examining the heatmap, it is possible to identify variables with high positive or negative correlations, which may indicate redundancy or collinearity. These variables can then be removed from the dataset, if necessary, using the code provided in the previous answer.

**59. Label Encoding cat\_vals = ['country', 'content\_rating', 'director\_name', 'actor\_1\_name', 'actor\_2\_name'] df[cat\_vals] = df[cat\_vals].apply(LabelEncoder().fit\_transform)**

The code provided applies label encoding to the categorical variables in the list **cat\_vals** in the dataframe **df**. Label encoding is a technique that assigns unique integers to each category in a categorical variable, allowing them to be represented as numerical values. label encoding may introduce ordinal relationships between categories that do not exist, which can lead to biased models. It is important to carefully consider the nature of the data and the specific use case before applying label encoding or any other encoding technique.

**62. log Transformation**

Log transformation is a data preprocessing technique that is used to normalize the distribution of skewed or non-normal data. It is often used in statistical analysis and machine learning to improve the accuracy and performance of models that assume normality of the data.

**63. Skewed features** are variables in a dataset whose distribution is not symmetric and is skewed towards one tail of the distribution. Skewness is a measure of the degree of asymmetry of a probability distribution.

**64. Applying log transformation on the skewed features**

After applying the log transformation, you should check the distribution of the transformed features to ensure that they are approximately normally distributed.

**65. Checking the changes in the distribution of data after applying log transformation.**

in this code, the seaborn.histplot() function is used to plot the distribution of the variable x before and after log transformation. The kde=False parameter is used to turn off the kernel density estimation plot, which can help emphasize the shape of the distribution.

After applying log transformation, the distribution of the variable should be closer to a normal distribution, with less skewness and a more symmetrical shape. This can help improve the performance of machine learning models that assume normality of the data.

### **66. Split Data**

After preprocessing the dataset, the next step is to split the data into training and testing sets. Here's an example code snippet that demonstrates how to split the data into 70% training and 30% testing sets:

After splitting the data, you can use the **X\_train**, **X\_test**, **y\_train**, and **y\_test** variables to train and evaluate machine learning models on the dataset.

**70. Machine Learning**

**Linear Regression**

Linear regression is a type of supervised learning algorithm used to model the relationship between a dependent variable **y** and one or more independent variables **X**. In the case of predicting the **imdb\_score** of a movie, we can use linear regression to model the relationship between the movie features (e.g., budget, runtime, etc.) and the target variable **imdb\_score**.

After training and evaluating the linear regression model, you can use it to make predictions on new data by calling the **predict()** method on the model object.

**71. Decision Tree Regressor**

Decision tree regression is another supervised learning algorithm that can be used to predict the **imdb\_score** of a movie based on its features. A decision tree is a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. In a decision tree regression model, each internal node represents a test on a feature, each branch represents the outcome of the test, and each leaf node represents a prediction of the target variable.

After training and evaluating the decision tree regression model, you can use it to make predictions on new data by calling the **predict()** method on the model object.

**72. Random Forest Regressor**

Random forest regression is an extension of decision tree regression that involves constructing multiple decision trees and averaging their predictions to improve generalization and reduce overfitting. In a random forest regression model, each tree is trained on a different random subset of the training data, and a random subset of features is selected at each node of the tree to split on.

After training and evaluating the random forest regression model, you can use it to make predictions on new data by calling the **predict()** method on the model object.