

ViewCount Visionary: A Data-Driven Approach to Forecasting YouTube Video Views

In today's world, social media has become a crucial part of our lives. One such platform that has gained immense popularity over the years is youtube. It is a platform where content creators can share their videos with a global audience, and advertisers can promote their products and services through video ads. Advertisers pay content creators based on the number of adviews and clocks their ads receive.

In recent years, advertisers have been looking for ways to estimate the number of adviews based on other metrics like comments, likes, dislikes, duration, year and category of the video. This is where machine learning comes in. by training various regression models on the available data, we can predict the number of adviews that an advertisement is likely to receive.

Catering to the problem stated above, we have developed a model which can predict the number of adviews given the specified inputs. This model can be used by advertisement agencies on deciding whether or not give a particular content creator on youtube to gove the contract for a particular ad or not based on the number of ad-views predicted by our model.

Aim:

The aim of this project is to develop, evaluate and deploy the best performing regression model to predict the number of adviews for Youtube advertisements based on their engagement metrics using Flask Framework.

Purpose of doing this Project:

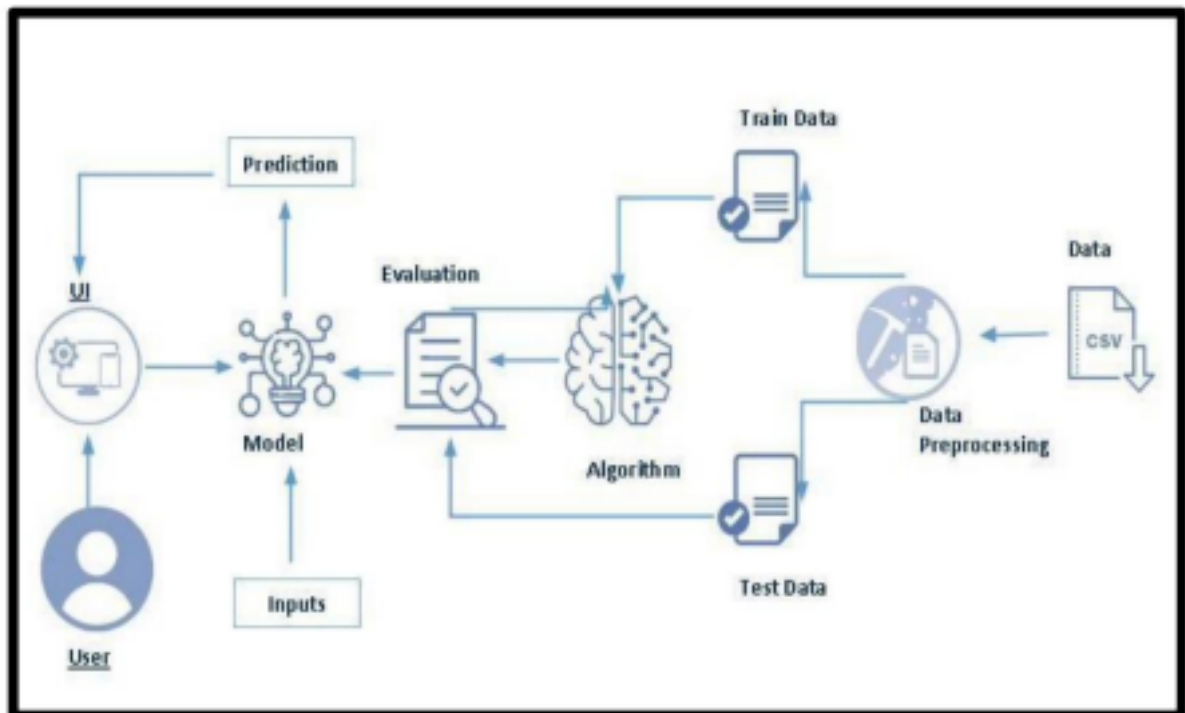
The purpose of this project is to develop and evaluate various regression models to accurately predict the number of adviews for Youtube advertisements based on the engagement metrics. Additionally, the project aims to deploy the best performing model using the Flask framework. Allowing advertisers and content creators to input relevant metrics and obtain an estimated number of ad-views. This can provide valuable insights to stakeholders, helping them to make informed decisions and improve their marketing strategies.

Technical Architecture:

In this architecture, a user inputs the relevant engagement metrics such as comments, likes, dislikes, duration, year and category of the video. The Flask application is responsible for handling the request and returning the predicted number of ad-views based on the inputs metrics.

The Flask application then passes the inputs metrics to the trained regression model, which processes the input data and returns a predicted number of ad-views based on the learned relationship between the input metrics and ad-views.

Once the predicted number of ad-views is calculated, the Flask application returns this information to the user's browser, where it is displayed on the user interface.



Project Flow:

- User is shown the Home page. The user will browse through Home page and enter the specified engagement metrics.
- After clicking the Predict button the user will be directed to the Results page where the model will analyse the inputs given by the user and showcase the prediction of the most estimated number of ad-views.

To accomplish this we have to complete all the activities listed below:

- Define problem / Problem understanding
 - Specify the business problem
 - Business Requirements
 - Literature Survey
 - Social or Business Impact
- Data Collection and Preparation
 - Collect the dataset
 - Data Preparation
- Exploratory Data Analysis
 - Descriptive statistical
 - Visual Analysis
- Model Building
 - Creating a function for evaluation
 - Training and testing the Models using multiple algorithms
- Performance Testing & Hyperparameter Tuning
 - Testing model with multiple evaluation metrics
 - Comparing model accuracy before & after applying hyperparameter tuning
 - Comparing model accuracy for different number of features.
 - Building model with appropriate features.
- Model Deployment
 - Save the best model
 - Integrate with Web Framework

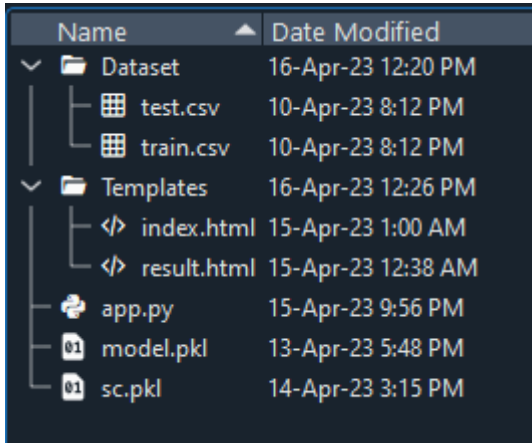
Prior Knowledge:

You must have the prior knowledge of the following topics to complete this project.

- ML Concepts:
 - Supervised learning: <https://www.javatpoint.com/supervised-machine-learning>
 - Linear Regression: <https://www.javatpoint.com/linear-regression-in-machine-learning>
 - SVM: <https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm>
 - Decision tree: <https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm>
 - Random forest: <https://www.javatpoint.com/machine-learning-random-forest-algorithm>
 - Evaluation metrics: <https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/>
 - Regularisation: <https://www.javatpoint.com/regularization-in-machine-learning>
- Flask Basics: https://www.youtube.com/watch?v=lj4I_CvBnt0

Project Structure:

Create project folder which contains files as shown below:



Name	Date Modified
Dataset	16-Apr-23 12:20 PM
test.csv	10-Apr-23 8:12 PM
train.csv	10-Apr-23 8:12 PM
Templates	16-Apr-23 12:26 PM
index.html	15-Apr-23 1:00 AM
result.html	15-Apr-23 12:38 AM
app.py	15-Apr-23 9:56 PM
model.pkl	13-Apr-23 5:48 PM
sc.pkl	14-Apr-23 3:15 PM

- The data obtained is in two csv files, one for training and another for testing.
- We are building a Flask application which will require the html files to be stored in the templates folder.
- app.py file is used for routing purposes using scripting.
- model.pkl is the saved model. This will further be used in the Flask integration.
- sc.pkl is saved scalar function. This will be used while predicting the ad-views.

Milestone 1: Define Problem/ Problem Understanding

Activity 1: Specify the Business Problem

The business problem in this scenario is to accurately predict the number of views a YouTube video will receive, which is crucial for advertisers to make informed decisions about which videos to sponsor and for content creators to optimize their content for maximum views and revenue. The goal is to develop a data-driven approach to forecasting YouTube video views that can be used by both advertisers and content creators to maximize their return on investment.

Activity 2: Business Requirements

Ad-View prediction can have various different business requirements, based on the specified goal. Few of the potential requirements are stated below-

1. Develop and evaluate various regression models to accurately predict the number of adviews for Youtube advertisements.
2. Choose the best performing model to predict the number of adviews.
3. Refine and clean the data before feeding it into the algorithms for better results.
4. Deploy the best performing model using the Flask framework.
5. Allow advertisers and content creators to input relevant metrics and obtain an estimated number of ad-views.
6. Provide valuable insights to stakeholders to make informed decisions and improve their marketing strategies.

- **Accurate and reliable information:** The regression models used for predicting adviews should be accurate and reliable since any false information can have severe consequences. The right symptoms and engagement metrics should be linked to the right adviews for providing the most precise output.
- **Trust:** The model should be designed in such a way that it develops trust among the users, especially the advertisers and content creators, who rely on the predicted adviews for their marketing strategies.
- **Compliance:** The model should comply with all the relevant laws and regulations set by the Central Drug Standard Control Organization, Ministry of Health, etc.
- **User-friendly interface:** The model should have a user-friendly interface to make it easy for users to input relevant metrics and obtain an estimated number of adviews.

Activity 3: Literature Survey

A literature survey for ad-view count would involve a comprehensive review of existing research and studies related to YouTube video views forecasting, regression models, and data-driven approaches. This would involve a search for relevant publications, articles, and academic papers on the topic, as well as an analysis of the various techniques, models, and algorithms used in previous research. The literature survey would also involve identifying gaps in existing research and potential areas for further exploration and improvement.

Activity 4: Social or Business Impact.

Social Impact:

The social impact of the ViewCount Visionary project could be significant. By accurately predicting the number of adviews for YouTube videos, advertisers and content creators can make more informed decisions and improve their marketing strategies, which could potentially increase revenue and profits. This could have a positive impact on the economy and job creation, as well as provide more opportunities for businesses to reach their target audiences. Additionally, by improving the accuracy of adviews estimation, it could reduce the potential for fraudulent or inaccurate reporting, increasing transparency and trust in the advertising industry.

Business Impact:

It can provide valuable insights to advertisers and content creators about the expected number of adviews for their videos. By accurately predicting adviews, advertisers can optimize their marketing strategies and maximize their return on investment. Content creators can also use this information to understand which videos are performing well and make data-driven decisions about the type of content they produce. This project can ultimately lead to improved revenue generation for both advertisers and content creators, making it a valuable tool for the YouTube ecosystem.

Milestone 2: Data Collection and Preparation:

Machine Learning depends heavily on data. It is the most crucial part aspect that makes algorithm training possible. So, this section guides on how to download dataset.

Activity 1: Collect the dataset

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

In this project we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

Link: <https://www.kaggle.com/datasets/sidharth178/youtube-adview-dataset>

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualisation techniques and some analysing techniques.

Note: There are a number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

Activity 1.1: Importing the Libraries Importing Necessary Libraries

```
In [2]: import numpy as np
import pandas as pd
import matplotlib.cm as cm
import matplotlib.pyplot as plt
import tensorflow as tf
import re
```

Activity 1.2: Read the Dataset

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas.

In pandas we have a function called read_csv() to read the dataset. As a parameter we have to give the directory of the csv file.

Reading Train.csv and Test.csv file

```
In [3]: data_train = pd.read_excel('C:/Users/kamya/Downloads/train.xlsx')
data_test = pd.read_excel('C:/Users/kamya/Downloads/test.xlsx')
```

```
In [4]: data_train.head()
```

Out[4]:

	vidid	adview	views	likes	dislikes	comment	published	duration	category
0	VID_18655	40.0	1031602.0	8523.0	363.0	1095.0	2016-09-14	PT7M37S	F
1	VID_14135	2.0	1707.0	56.0	2.0	6.0	2016-10-01	PT9M30S	D
2	VID_2187	1.0	2023.0	25.0	0.0	2.0	2016-07-02	PT2M16S	C
3	VID_23096	6.0	620860.0	777.0	161.0	153.0	2016-07-27	PT4M22S	H
4	VID_10175	1.0	666.0	1.0	0.0	0.0	2016-06-29	PT31S	D

```
In [5]: data_test.head()
```

Out[5]:

	vidid	views	likes	dislikes	comment	published	duration	category
0	VID_1054	440238.0	6153.0	218.0	1377.0	2017-02-18	PT7M29S	B
1	VID_18629	1040132.0	8171.0	340.0	1047.0	2016-06-28	PT6M29S	F
2	VID_13967	28534.0	31.0	11.0	1.0	2014-03-10	PT37M54S	D
3	VID_19442	1316715.0	2284.0	250.0	274.0	2010-06-05	PT9M55S	G

As we have two datasets, one for training and other for testing we will import both the csv files.

Activity 2: Data Preparation

As we have understood how the data is, let us preprocess the collected data.

The Machine Learning model cannot be trained on the imported data directly. The dataset might have randomness, we might have to clean the dataset and bring it in the right form. This activity involves the following steps:

- Removing Redundant Columns
- Handling Missing Values

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

Activity 2.1: Removing Redundant Data

Removing character "F" present in data

```
In [10]: data_train=data_train[data_train.views!='F']
data_train=data_train[data_train.likes!='F']
data_train=data_train[data_train.dislikes!='F']
data_train=data_train[data_train.comment!='F']
data_train.head()
```

Out[10]:

	vidid	adview	views	likes	dislikes	comment	published	duration	category
0	VID_18655	40.0	1031602.0	8523.0	363.0	1095.0	2016-09-14	PT7M37S	F
1	VID_14135	2.0	1707.0	56.0	2.0	6.0	2016-10-01	PT9M30S	D
2	VID_2187	1.0	2023.0	25.0	0.0	2.0	2016-07-02	PT2M16S	C
3	VID_23096	6.0	620860.0	777.0	161.0	153.0	2016-07-27	PT4M22S	H
4	VID_10175	1.0	666.0	1.0	0.0	0.0	2016-06-29	PT31S	D

```
In [11]: data_test=data_test[data_test.views!='F']
data_test=data_test[data_test.likes!='F']
data_test=data_test[data_test.dislikes!='F']
data_test=data_test[data_test.comment!='F']
data_test.head()
```

Out[11]:

	vidid	views	likes	dislikes	comment	published	duration	category
0	VID_1054	440238.0	6153.0	218.0	1377.0	2017-02-18	PT7M29S	B

Removing the unwanted character 'F' from data.

Activity 2.2: Converting Categorical data to Numerical data

Assigning each category a number for Category feature

```
In [12]: category={'A': 1, 'B':2, 'C':3, 'D':4, 'E':5, 'F':6, 'G':7, 'H':8}
data_train["category"]=data_train["category"].map(category)
data_train.head()
```

Out[12]:

	vidid	adview	views	likes	dislikes	comment	published	duration	category
0	VID_18655	40.0	1031602.0	8523.0	363.0	1095.0	2016-09-14	PT7M37S	6
1	VID_14135	2.0	1707.0	56.0	2.0	6.0	2016-10-01	PT9M30S	4
2	VID_2187	1.0	2023.0	25.0	0.0	2.0	2016-07-02	PT2M16S	3
3	VID_23096	6.0	620860.0	777.0	161.0	153.0	2016-07-27	PT4M22S	8
4	VID_10175	1.0	666.0	1.0	0.0	0.0	2016-06-29	PT31S	4

```
In [13]: category={'A': 1, 'B':2, 'C':3, 'D':4, 'E':5, 'F':6, 'G':7, 'H':8}
data_test["category"]=data_test["category"].map(category)
data_test.head()
```

Out[13]:

	vidid	views	likes	dislikes	comment	published	duration	category
0	VID_1054	440238.0	6153.0	218.0	1377.0	2017-02-18	PT7M29S	2
1	VID_18629	1040132.0	8171.0	340.0	1047.0	2016-06-28	PT6M29S	6
2	VID_13967	28534.0	31.0	11.0	1.0	2014-03-10	PT37M54S	4

Assigned each category in the category column a number.

Activity 2.3:

Converting values to integers for views, likes, comments, dislikes and adview

```
In [14]: data_train["views"] = pd.to_numeric(data_train["views"])
data_train["comment"] = pd.to_numeric(data_train["comment"])
data_train["likes"] = pd.to_numeric(data_train["likes"])
data_train["dislikes"] = pd.to_numeric(data_train["dislikes"])
data_train["adview"] = pd.to_numeric(data_train["adview"])
column_vidid=data_train['vidid']
```

```
In [15]: data_test["views"] = pd.to_numeric(data_test["views"])
data_test["comment"] = pd.to_numeric(data_test["comment"])
data_test["likes"] = pd.to_numeric(data_test["likes"])
data_test["dislikes"] = pd.to_numeric(data_test["dislikes"])
column_vidid1=data_test['vidid']
```

The code is converting the following columns in both the training and test datasets to numeric data type: views, comment, likes, dislikes, and adview.

Then, it is storing the vidid column from the training and test datasets in separate variables for later use.

This data preprocessing step is important for improving the accuracy of the machine learning models that will be used to predict adviews based on engagement metrics.

Activity 2.3: Converting String to Integer format

Converting Time_in_sec for duration

```
In [16]: def getSeconds(x):
pat = re.compile('PT(?:\d+)H(?:\d+)M(?:\d+)S?')
m = pat.match(str(x))
return (int(m.group(1))*60*60 if m.group(1) else 0) + (int(m.group(2))*60 if m.group(2) else 0) + (int(m.group(3)) if m.g

data_train['duration'] = data_train['duration'].apply(getSeconds)
data_test['duration'] = data_test['duration'].apply(getSeconds)
```

This code defines a function `getSeconds(x)` that takes in a string argument in the format "PT#H#M#S" where "#" represents a number of hours, minutes, and seconds, respectively. The function extracts the values of hours, minutes, and seconds from the string using regular expressions, converts them to seconds, and returns the total duration in seconds.

This function is then applied to the 'duration' column of both the training and test datasets using the `apply()` method. This converts the 'duration' column from string format to integer format representing the total duration of the video in seconds.

Converting date to Year published for published

```
In [17]: def getYear(a):
pat = re.compile('(?:\d+)-(?:\d+)-(?:\d+)-?')
n = pat.match(str(a))
return (int(n.group(1)) if n.group(1) else 0)
data_train['published'] = data_train['published'].apply(getYear)
data_test['published'] = data_test['published'].apply(getYear)
```

This code defines a function called `getYear` which takes a string argument representing a date and extracts the year from it. It does this by using regular expressions to match the year in the format "YYYY-" and returns the year as an integer. If the year is not present in the string, it returns 0.

The `apply` function is then used to apply this function to the "published" column of both the training and testing datasets. This converts the "published" column from a string to an integer representing the year.

Milestone 3: Exploratory Data Analysis

Activity 1: Descriptive Statistical

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas has a worthy function called `describe`. With this `describe` function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features.

```
In [43]: data_train.describe()
```

Out[43]:

	views	likes	dislikes	comment	published	duration	category
count	1.463600e+04	14636.000000	14636.000000	14636.000000	14636.000000	14636.000000	14636.000000
mean	7.107934e+05	2784.093946	254.150724	409.035597	9.437346	622.944999	4.607065
std	2.731062e+06	8936.295816	1029.257991	1511.180179	1.770455	710.910074	1.576242
min	4.900000e+01	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000
25%	3.151425e+04	88.000000	7.000000	7.000000	8.000000	183.000000	4.000000
50%	1.586610e+05	450.000000	38.000000	46.000000	10.000000	321.000000	4.000000
75%	5.829575e+05	1861.500000	166.250000	224.000000	11.000000	719.000000	6.000000
max	1.380479e+08	283824.000000	49449.000000	75045.000000	12.000000	3077.000000	8.000000

Activity 2: Visualisation

Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

Activity 2.1:

Visualization

```
# Individual Plots
plt.hist(data_train["category"])
plt.show()
plt.plot(data_train["adview"])
plt.show()
# Remove videos with adview greater than 2000000 as outlier
data_train = data_train[data_train["adview"] < 2000000]
# Heatmap
import seaborn as sns
f, ax = plt.subplots(figsize=(10, 8))
corr = data_train.corr()
sns.heatmap(corr, mask = np.zeros_like(corr, dtype=np.bool), cmap=sns.diverging_palette(220, 10, as_cmap=True),
            square=True, ax=ax, annot=True)
plt.show()
```

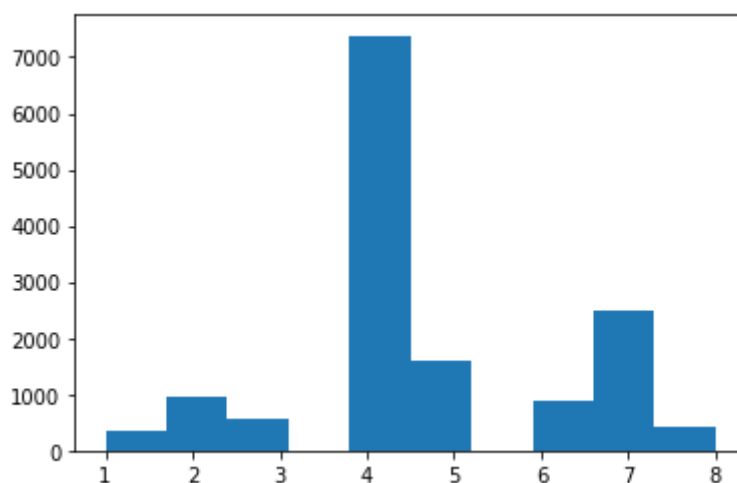
This code is plotting two individual plots for a dataset named "data_train".

The first plot is a histogram using the "plt.hist" function, which shows the distribution of the "category" variable in the dataset.

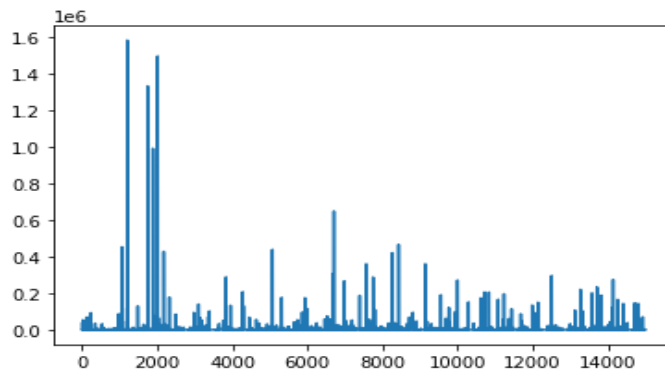
The second plot is a line plot using the "plt.plot" function, which shows the trend of the "adview" variable in the dataset.

Additionally, the code removes any rows in the dataset where the "adview" variable is greater than 2000000, which are considered outliers. This is done using the pandas boolean indexing syntax, "data_train[data_train["adview"] < 2000000]".

Data plot of Category column.



Data plot of Category column.



Data plot of Adview column

Activity 2.2:

```
# Heatmap
import seaborn as sns
f, ax = plt.subplots(figsize=(10, 8))
corr = data_train.corr()
sns.heatmap(corr, mask = np.zeros_like(corr, dtype=np.bool), cmap=sns.diverging_palette(220, 10, as_cmap=True),
            square=True, ax=ax, annot=True)
plt.show()
```

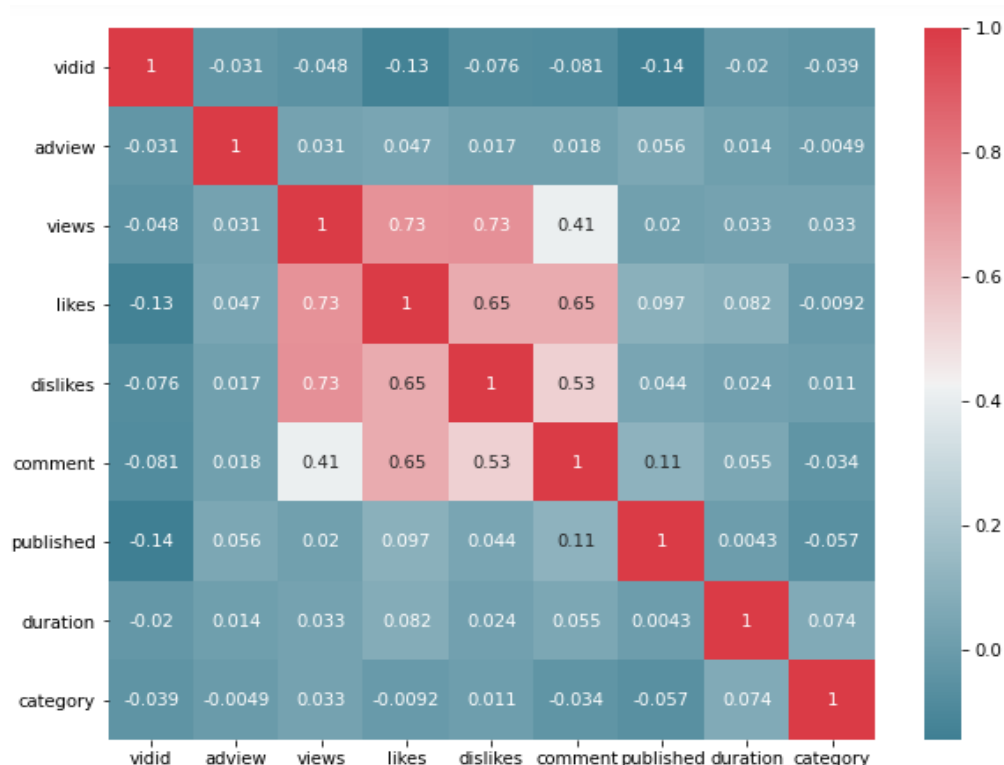
This code generates a heatmap of the correlation matrix for the "data_train" dataset using the seaborn library.

The "plt.subplots" function creates a figure and a set of subplots with a specified size.

The "data_train.corr()" function calculates the correlation coefficients between all numerical variables in the dataset.

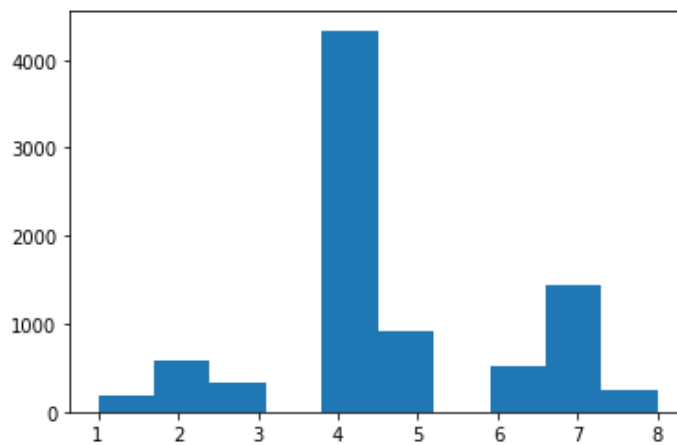
The "sns.heatmap" function creates a heatmap using the correlation matrix as input. The "mask" argument is used to mask out the upper triangle of the heatmap since it's a symmetric matrix. The "cmap" argument specifies the color palette for the heatmap. The "square" argument makes sure the heatmap is square. The "annot" argument displays the correlation values in each cell of the heatmap.

Finally, "plt.show()" is used to display the heatmap.



Activity 2.3:

```
# Individual Plots
plt.hist(data_test["category"])
plt.show()
```



Activity 2.4: Split data into training and testing data

Splitting Data

```
In [25]: Y_train = pd.DataFrame(data = data_train.iloc[:, 1].values, columns = ['target'])
test_y = pd.DataFrame(data_test.iloc[:, 0].values, columns=['vidid'])
data_train=data_train.drop(["adview"],axis=1)
data_train=data_train.drop(["vidid"],axis=1)
data_test = data_test.drop(['vidid'], axis=1)
data_train.head()
```

Out[25]:

	views	likes	dislikes	comment	published	duration	category
0	1031602.0	8523.0	363.0	1095.0	11	453	6
1	1707.0	56.0	2.0	6.0	11	566	4
2	2023.0	25.0	0.0	2.0	11	132	3
3	620860.0	777.0	161.0	153.0	11	258	8
4	666.0	1.0	0.0	0.0	11	27	4

This code performs the following tasks.

- Creates a new DataFrame called "Y_train" using the "pd.DataFrame" function. The values in the "target" column of the "data_train" DataFrame are selected using the "iloc" function and then converted into a new DataFrame called "Y_train". The "columns" argument is used to rename the column to "target".
- Creates a new DataFrame called "test_y" using the "pd.DataFrame" function. The values in the "vidid" column of the "data_test" DataFrame are selected using the "iloc" function and then converted into a new DataFrame called "test_y". The "columns" argument is used to rename the column to "vidid".
- Removes the "adview" and "vidid" columns from the "data_train" DataFrame using the "drop" function with the "axis=1" argument. The modified DataFrame is stored back in the "data_train" variable.
- Removes the "vidid" column from the "data_test" DataFrame using the "drop" function with the "axis=1" argument.
- Displays the first five rows of the modified "data_train" DataFrame using the "head()" function to verify the changes.

Milestone 4: Model Building

Activity 1: Normalising

We need to normalize the data to ensure that all features have the same scale, to prevent the dominance of features with higher values, to improve the performance and convergence of machine learning algorithms, and to reduce the impact of outliers and improve the numerical stability of the optimization process.

Normalising Data

```
In [27]: > from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train=scaler.fit_transform(X_train)
X_test=scaler.fit_transform(X_test)
data_test = scaler.transform(data_test)
X_train.mean()
```

```
Out[27]: 0.21883359153168205
```

Activity 2: Creating a function for model evaluation

Evaluation Metrics

```
> from sklearn import metrics
def print_error(X_test, y_test, model_name):
    prediction = model_name.predict(X_test)
    print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, prediction))
    print('Mean Squared Error:', metrics.mean_squared_error(y_test, prediction))
    print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, prediction)))
```

This code defines a function called "print_error" that takes three arguments: "X_test", "y_test", and "model_name". The function uses the "predict" method of the "model_name" object to generate predictions for the "X_test" input, and then calculates and prints three evaluation metrics using functions from the "sklearn.metrics" module: mean absolute error (MAE), mean squared error (MSE), and root mean squared error (RMSE).

The MAE represents the average absolute difference between the predicted and actual values, the MSE represents the average squared difference between the predicted and actual values, and the RMSE is the square root of the MSE. These metrics provide a measure of the accuracy and magnitude of errors in the model's predictions. The "print_error" function is a useful tool for evaluating the performance of a machine learning model on a test set and can help identify areas for improvement.

Mean Absolute Error (MAE): This is the average absolute difference between the predicted values and the actual values. MAE is a measure of how far off the predictions are from the actual values. A lower MAE indicates that the model is better at predicting the target variable.

Mean Squared Error (MSE): This is the average of the squared differences between the predicted values and the actual values. MSE penalizes large errors more than small errors. A lower MSE also indicates a better-performing model.

Root Mean Squared Error (RMSE): This is the square root of the MSE. RMSE is another measure of how far off the predictions are from the actual values. It is used to put the error metric back into the same units as the target variable.

Activity 3: Training and testing the models using multiple algorithms

Now that we have clean data, a function for evaluation it is time to build models to train the data. For this project we will be using 7 different regression algorithms to build our models. The best model will be used for prediction.

Activity 3.1: Linear Regression

Linear regression is a statistical method for modeling the relationship between a dependent variable and one or more independent variables. It assumes a linear relationship between the variables and estimates the coefficients of the linear equation to make predictions.

Linear Regression

```
from sklearn import linear_model
linear_regression = linear_model.LinearRegression()
linear_regression.fit(X_train, y_train)
print_error(X_test, y_test, linear_regression)
```

```
Mean Absolute Error: 3713.223309547266
Mean Squared Error: 835785877.9329115
Root Mean Squared Error: 28909.961569205025
```

The Mean Absolute Error (MAE) indicates the average magnitude of the errors between the predicted and actual values, and the MAE of 3713 suggests that the model predictions are, on average, off by about 3713. The Mean Squared Error (MSE) measures the average squared difference between the predicted and actual values, and the RMSE (Root Mean Squared Error) represents the square root of the MSE, indicating that the model has an average prediction error of about 28910.

Activity 3.2: Support Vector Regressor

Support Vector Regressor (SVR) is a regression algorithm that uses Support Vector Machine (SVM) principles to build a model. It finds a line or a hyperplane that has the maximum possible margin from the target variable to make predictions on new data.

Support Vector Regressor

```
from sklearn.svm import SVR
supportvector_regressor = SVR()
supportvector_regressor.fit(X_train, y_train)
print_error(X_test, y_test, supportvector_regressor)
```

```
Mean Absolute Error: 1696.9578659483836
Mean Squared Error: 833685959.9169687
Root Mean Squared Error: 28873.620485089305
```

The support vector regressor model has a lower Mean Absolute Error and Mean Squared Error than the linear regression model, indicating better performance. The Root Mean Squared Error is also lower, indicating that the model's predictions are more accurate.

Activity 2.3: Decision Tree Regressor

Decision Tree Regressor is a regression algorithm that works by recursively partitioning the data into smaller subsets based on the most significant attribute, creating a tree-like model of decisions and their possible consequences. The model predicts the target variable by traversing the tree based on the attribute values of the input data.

Decision Tree Regressor

```
from sklearn.tree import DecisionTreeRegressor
decision_tree = DecisionTreeRegressor()
decision_tree.fit(X_train, y_train)
print_error(X_test, y_test, decision_tree)
```

```
Mean Absolute Error: 2288.718066939891
Mean Squared Error: 943114919.456711
Root Mean Squared Error: 30710.17615476523
```

The decision tree regressor has a mean absolute error of 2288.71 and a root mean squared error of 30710.18. These values indicate the average difference between the actual and predicted values. The performance of the model can be further improved by tuning the hyperparameters.

Activity 2.4: Random Forest Regressor

Random Forest Regressor is an ensemble algorithm that combines multiple decision trees to make predictions on new data. It randomly samples a subset of features and data points to train each tree, reducing overfitting and increasing accuracy. The final prediction is obtained by averaging the predictions of all the trees.

Random Forest Regressor

```
from sklearn.ensemble import RandomForestRegressor
n_estimators = 200
max_depth = 25
min_samples_split=15
min_samples_leaf=2
random_forest = RandomForestRegressor(n_estimators = n_estimators, max_depth = max_depth
                                     , min_samples_split=min_samples_split,min_samples_leaf=min_samples_leaf)
random_forest.fit(X_train,y_train)
print_error(X_test,y_test, random_forest)
```

```
Mean Absolute Error: 3620.769931179114
Mean Squared Error: 903361165.193317
Root Mean Squared Error: 30055.967214403816
```

The random forest regressor model has an average error of 3620.77 and predicts ad views with a root mean squared error of 30055.97. This indicates that the model is not very accurate in predicting ad views.

Activity 2.4: Ridge Regression

Ridge Regression is a linear regression algorithm that adds a regularization term to the cost function to reduce overfitting. The regularization term penalizes large weights and biases, forcing the model to generalize better to new data. This algorithm is commonly used when there are many correlated input variables.

Ridge Regression

```
from sklearn.linear_model import Ridge
ridge = Ridge(alpha=1)
ridge.fit(X_train, y_train)
print_error(X_test, y_test, ridge)
```

```
Mean Absolute Error: 3679.727863220872
Mean Squared Error: 834282391.302759
Root Mean Squared Error: 28883.946948136418
```

The root mean squared error of 28,883 indicates that the average difference between the predicted and actual values is around 28,883. This indicates a relatively high error rate for the model, which can be improved by adjusting the hyperparameters.

Activity 2.4: Lasso Regression

Lasso Regression is a linear regression algorithm that adds a regularization term to the cost function to reduce overfitting. The regularization term includes the absolute values of the weights, which can set some of them to zero, effectively performing feature selection. This algorithm is commonly used when there are many irrelevant input variables.

Lasso Regression

```
from sklearn.linear_model import Lasso
lasso = Lasso(alpha=0.1)
lasso.fit(X_train, y_train)
print_error(X_test, y_test, lasso)
```

```
Mean Absolute Error: 3711.176087027305
Mean Squared Error: 835720353.2859067
Root Mean Squared Error: 28908.828293203216
```

For Lasso Regression, the Mean Absolute Error (MAE) is 3711.18, the Mean Squared Error (MSE) is 835720353.29, and the Root Mean Squared Error (RMSE) is 28908.83. These evaluation metrics are used to measure the accuracy of the model. The lower the values of MAE, MSE, and RMSE, the better the model is performing.

Activity 2.4:XG boost

XGBoost (Extreme Gradient Boosting) is a gradient boosting algorithm that uses decision trees as base models. It trains models iteratively, using the gradient descent algorithm to minimize a loss function. The algorithm employs regularization to prevent overfitting and provides a fast and efficient implementation of gradient boosting that has achieved state-of-the-art performance on many machine learning tasks.

XGBoost

```
import xgboost as xgb
from sklearn.datasets import load_boston
xgb_model = xgb.XGBRegressor(objective="reg:squarederror", random_state=42)
xgb_model.fit(X_train, y_train)
print_error(X_test,y_test, xgb_model)
```

```
Mean Absolute Error: 3586.5989905268443
Mean Squared Error: 801097300.0120089
Root Mean Squared Error: 28303.662307411898
```

The XGBoost model has an MAE of 3586.6, MSE of 801097300, and RMSE of 28303.66, indicating good performance in predicting the target variable. The lower the values, the better the model's accuracy in making predictions.

Milestone 5 : Performance Testing

Activity 1: Comparing all the Models.

Based on the given metrics, the SVR model seems to be performing the best among all the given models with the lowest MAE, MSE, and RMSE values. The linear regression, decision tree, and random forest models also have relatively low MAE and RMSE values, but the SVR model has the lowest values of all the metrics. The XGBoost model has the highest values of all the metrics, indicating that it's performing the worst among all the given models. However, it's important to note that the model comparison should also consider other factors such as the complexity of the model, training and testing time, and interpretability, among others.

Predicting

```
preds = model.predict(data_test)
preds = preds.astype(np.int16)
preds = preds.reshape(1, -1)[0].tolist()
```

This code runs predictions on a test dataset using a machine learning model called model. The predicted values are then converted to 16-bit integers using NumPy's astype method. The predicted values are then reshaped to a one-dimensional array and converted to a Python list using the reshape and tolist methods, respectively. This post-processing of the predicted values may be useful for further analysis or evaluation.

Milestone 6: Model Deployment

Activity 1: Save and load the best model

Saving the best model after comparing its performance using different evaluation metrics means selecting the model with the highest performance and saving its weights and configuration. This can be useful in avoiding the need to retrain the model every time it is needed and also to be able to use it in the future.

After checking the performance, we decide to save the Support Vector Regressor model.

```
➤ import joblib
  joblib.dump(supportvector_regressor, "model.pkl")

In [ ]: ['model.pkl']

➤ model = joblib.load('model.pkl')
```

We save the model using the pickle library into a file named model.pkl

Activity 2: Save and load the Scaler function

```
➤ import joblib
  joblib.dump(scaler, "sc.pkl")

In [ ]: ['sc.pkl']

➤ scaling = joblib.load('sc.pkl')
```

This code uses the joblib module to perform serialization and deserialization of a scaler object. The joblib.dump function saves the scaler object to a file called sc.pkl. This file can be used to restore the scaler object later on. The joblib.load function loads the saved scaler object from the sc.pkl file, which is then stored in the scaling variable. This process of saving and loading a machine learning model or preprocessing object using joblib can be useful for reusing the same model or preprocessing pipeline across different projects or on different machines.

Activity 2: Integrate with Web Framework

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the users where he has to enter the values for predictions. The entered values are given to the saved model and prediction is showcased on the UI. This section has the following tasks

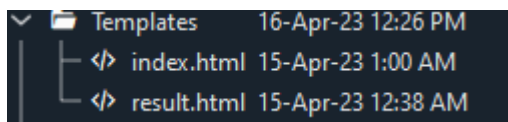
- Building HTML Pages
- Building server-side script
- Run the web application

Activity 2.1: Building HTML pages:

For this project we create two HTML files namely

- Index.html
- Results.html

And we will save them in the templates folder.



Activity 2.2: Build Python code

Create a new app.py file which will be stored in the Flask folder.

- Import the necessary Libraries.

```
from flask import Flask, render_template, request
import joblib
import numpy as np
```

This code uses the joblib module to load a saved machine learning model and a saved preprocessing object. The joblib.load function is used to load the model object from a file called model.pkl and the scaler object from a file called sc.pkl. These objects are then stored in the model and scaler variables, respectively. The loaded model object can be used to make predictions on new data and the loaded scaler object can be used to preprocess the data in the same way as it was done during training. This process of loading saved model and preprocessing objects can save time and resources when working on a new project or dataset.

```
# Load the trained model and scaler
model = joblib.load('model.pkl')
scaler = joblib.load('sc.pkl')
```

This code creates a new instance of a Flask web application using the Flask class from the Flask library. The __name__ argument specifies the name of the application's module or package.

```
app = Flask(__name__)
```

This code creates a Flask route for the root URL (/) of the application, and maps it to the home function. When a user accesses the root URL in their web browser, Flask calls the home function, which returns the result of rendering an HTML template called index.html using the render_template function. This allows dynamic content to be displayed on the webpage, as the placeholders within the index.html template can be filled with data from the application.

```
@app.route('/')
def home():
    return render_template('index.html')
```

This code defines a Flask route for the /predict URL, and maps it to the predict function. When a user submits a form with data using the POST method to the /predict URL, Flask calls the predict function. The function first extracts the user-submitted data from the form, and then creates an array of features using NumPy. The features are then scaled using the scaler object that was previously loaded. The scaled features are then passed to the loaded model object to obtain a prediction, which is rounded to two decimal places. The prediction is then passed to an HTML template called result.html using the render_template function, and displayed to the user as part of the response.

```
@app.route('/predict', methods=['POST'])
def predict():
    views = float(request.form['views'])
    likes = float(request.form['likes'])
    dislikes = float(request.form['dislikes'])
    comments = float(request.form['comments'])
    year = float(request.form['year'])
    duration = float(request.form['duration'])
    category = float(request.form['category'])

    features = np.array([[views, likes, dislikes, comments, year, duration, category]])
    features_scaled = scaler.transform(features)

    prediction = model.predict(features_scaled)
    output=round(prediction[0],2)

    return render_template('result.html', prediction_text=' {}'.format(prediction))
```

Main Function:

This code runs the Flask application if the script is being executed directly (i.e. not imported as a module in another script). The if __name__ == '__main__': line checks if the script is the main module being executed, and if so, runs the Flask application using the app.run() method. This method starts the Flask development server, allowing the application to be accessed via a web browser at the appropriate URL.

```
if __name__ == '__main__':
    app.run()
```

Activity 2.3: Run the Web Application

Our business name is 'OmniCom'.

When you run the "app.py" file this window will open in the console or output terminal. Copy the URL given in the form <http://127.0.0.1:5000> and paste it in the browser.

```
In [2]: runfile('C:/Users/kamya/OneDrive/Desktop/Ad-view prediction/
app.py', wdir='C:/Users/kamya/OneDrive/Desktop/Ad-view prediction')
* Serving Flask app "app" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a
  production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

When we paste the URL in a web browser, our index.html page will open. It contains various sections such as Number of views in video, Number of likes in video, Number of dislikes in video, Number of comments in video, Year of publish, Duration of video (in seconds) and Category of the video.

There is some information given on the web page about our model.

If you click on the Predict button on home page you will be redirected to the results.html page.

Our index.html looks as shown below:



The screenshot shows a web application for 'OmniCom'. The header features the 'OmniCom' logo. The main heading is 'View-Count Visionary'. Below this, a message states: 'Enter the details of your video below and we will predict the number of views it will receive!'. The form contains four input fields with labels: 'Number of views in video:', 'Number of likes in video:', 'Number of dislikes in video:', and 'Number of comments in video:'. At the bottom, a red footer bar contains the contact information: 'Contact us: 9873593560 Email: omnicom@gmail.com'.

Enter the values in this form to get the result.

Number of views in video:

3567

Number of likes in video:

8765

Number of dislikes in video:

6433

Number of comments in video:

678

Year of publish:

2020

Duration of video (in seconds):

68765

Category of the video:

Video Testimonials

Predict

When you as a client/user will enter some values in the input field and then hit on 'Predict' button you will be directed to the result page , where you can see the expected number of ad-views based onn the input values provided.

You will be redirected to the Results.html page once we click the Predict button.

Prediction Result

The estimated number of Ad-Views is: **[2.]**

The results say that based on Category provided and other engagement metrics provided the expected number is '2'. Therefore, a business it's not profitable to go for such videos.

Link to GitHub Repository:

<https://github.com/Kamya-Paliwal/Ad-ViewCount-A-Data-Driven-Approach>