SUMMER INTERNSHIP REPORT

(May – July 2024)

Akash K V
420PH5021
Final-year
Integrated MSc
kvakash180@gmail.com

Title of the Project:

YouTube Ad View Prediction Model

Supervisor: Mr. Kashish Kumar

Platform where Internship Work was Done: Internshipstudio

Area of Research:

Machine Learning, Data Analytics and Web Scraping

This research focuses on using machine learning and data analytics to predict YouTube adviews based on video metrics like views, likes, and comments. By employing regression models and web scraping with Python, the project aims to optimize advertising strategies and enhance decision-making in digital marketing.

Definition of the Problem:

The primary objective of this project was to predict the number of ad views on YouTube videos based on various video metrics, such as views, likes, dislikes, comments, published date, duration, and category. This prediction is crucial for YouTube advertisers, who pay content creators based on the number of ad views and clicks generated by their videos. Accurate ad view predictions enable advertisers to optimize their campaigns.

1. Import the datasets and libraries, check shape and datatype.

2. Visualise the dataset using plotting using heatmaps and plots. You can study data distributions for each attribute as well.

3. Clean the dataset by removing missing values and other things.

4. Transform attributes into numerical values and other necessary transformations

5. Normalise your data and split the data into training, validation and test set in the appropriate ratio.

6. Use linear regression, Support Vector Regressor for training and get errors.

7. Use Decision Tree Regressor and Random Forest Regressors.

8. Build an artificial neural network and train it with different layers and hyperparameters. Experiment a little. Use keras.

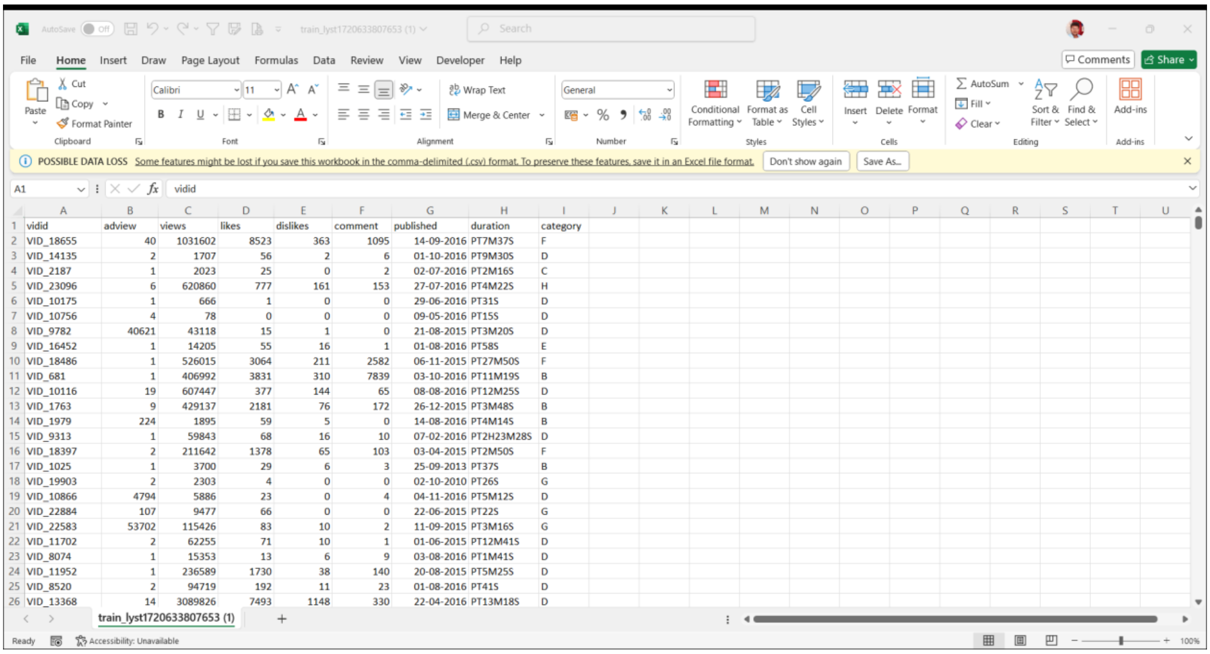
9. Pick the best model based on error as well as generalisation.

10.Save your model

Standard Data set for training a model

Training data

Data Description The file train.csv contains metrics and other details of about 15000 youtube videos. The metrics include number of views, likes, dislikes, comments and apart from that published date, duration and category are also included. The train.csv file also contains the metric number of adviews which is our target variable for prediction.



	vidid	adview	views	likes	dislikes	comment	published	duration	category
2	VID_18655	40	1031602	8523	363	1095	14-09-2016	PT7M37S	F
3	VID_14135	2	1707	56	2	6	01-10-2016	PT9M30S	D
4	VID_2187	1	2023	25	0	2	02-07-2016	PT2M16S	C
5	VID_23096	6	620860	777	161	153	27-07-2016	PT4M22S	H
6	VID_10175	1	666	1	0	0	29-06-2016	PT31S	D
7	VID_10756	4	78	0	0	0	09-05-2016	PT15S	D
8	VID_9782	40621	43118	15	1	0	21-08-2015	PT3M20S	D
9	VID_16452	1	14205	55	16	1	01-08-2016	PT58S	E
10	VID_18486	1	526015	3064	211	2582	06-11-2015	PT27M50S	F
11	VID_681	1	406992	3831	310	7839	03-10-2016	PT11M19S	B
12	VID_10116	19	607447	377	144	65	08-08-2016	PT12M25S	D
13	VID_1763	9	429137	2181	76	172	26-12-2015	PT3M48S	B
14	VID_1979	224	1895	59	5	0	14-08-2016	PT4M14S	B
15	VID_9313	1	59843	68	16	10	07-02-2016	PT2H23M28S	D
16	VID_18397	2	211642	1378	65	103	03-04-2015	PT2M50S	F
17	VID_1025	1	3700	29	6	3	25-09-2013	PT37S	B
18	VID_19903	2	2303	4	0	0	02-10-2010	PT26S	G
19	VID_10866	4794	5886	23	0	4	04-11-2016	PT5M12S	D
20	VID_22884	107	9477	66	0	0	22-06-2015	PT22S	G
21	VID_22583	53702	115426	83	10	2	11-09-2015	PT3M16S	G
22	VID_11702	2	62255	71	10	1	01-06-2015	PT12M41S	D
23	VID_8074	1	15353	13	6	9	03-08-2016	PT1M41S	D
24	VID_11952	1	236589	1730	38	140	20-08-2015	PT5M25S	D
25	VID_8520	2	94719	192	11	23	01-08-2016	PT41S	D
26	VID_13368	14	3089826	7493	1148	330	22-04-2016	PT13M18S	D

Methodology (Computational)::

1. **Data Import and Exploration:** The dataset was imported using Pandas, and initial exploratory analysis was conducted to understand the data shape, types, and distribution of each attribute.

```
import numpy as np
import pandas as pd
import matplotlib.cm as cn
import matplotlib.pyplot as plt

# Importing data
data_train = pd.read_csv("train.csv")
#finding the shape
data_train.shape
```

```
(14999, 9)
```

```
data_train.head()
```

	vidid	adview	views	likes	dislikes	comment	published	duration	category
0	VID_18655	40	1031602	8523	363	1095	2016-09-14	PT7M37S	F
1	VID_14135	2	1707	56	2	6	2016-10-01	PT9M30S	D
2	VID_2187	1	2023	25	0	2	2016-07-02	PT2M16S	C
3	VID_23096	6	620860	777	161	153	2016-07-27	PT4M22S	H
4	VID_10175	1	666	1	0	0	2016-06-29	PT31S	D

```
data_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14999 entries, 0 to 14998
Data columns (total 9 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   vidid      14999 non-null  object
 1   adview     14999 non-null  int64 
 2   views      14999 non-null  object
 3   likes      14999 non-null  object
 4   dislikes   14999 non-null  object
 5   comment    14999 non-null  object
 6   published  14999 non-null  object
 7   duration   14999 non-null  object
 8   category   14999 non-null  object
dtypes: int64(1), object(8)
memory usage: 1.0+ MB
```

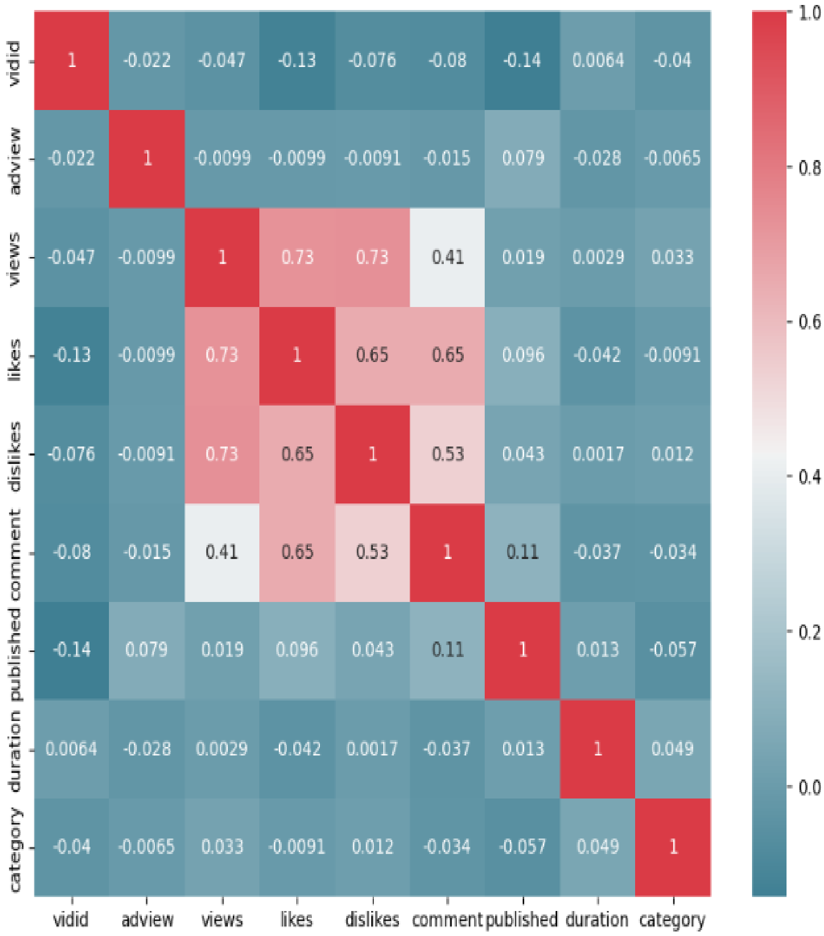
2. **Data Visualization:** Visualizations such as heatmaps and distribution plots were created using Matplotlib and Seaborn to study relationships between ad views. This step provided insights into correlations and patterns in the data.

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

data_train = data_train[data_train["adview"] < 200000]

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

plt.show()
```



3. **Data Cleaning and Preprocessing:** Missing values were handled, and non-numeric attributes (e.g., published date, duration) were transformed into numerical values. Data normalization was performed to standardize the scale of input features.

```
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()

# Convert values to integers for views, likes, comments, dislikes and
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']

# Endoding features like Category, Duration, Vidid
from sklearn.preprocessing import LabelEncoder
data_train['duration']=LabelEncoder().fit_transform(data_train['duration'])
data_train['vidid']=LabelEncoder().fit_transform(data_train['vidid'])
data_train['published']=LabelEncoder().fit_transform(data_train['published'])

data_train.head()
```

	vidid	adview	views	likes	dislikes	comment	published	duration	category
0	5912	40	1031602	8523	363	1095	2168	2925	6
1	2741	2	1707	56	2	6	2185	3040	4
2	8138	1	2023	25	0	2	2094	1863	3
3	9005	6	620860	777	161	153	2119	2546	8
4	122	1	666	1	0	0	2091	1963	4

4. Normalise your data and split the data into training, validation and test set in the appropriate ratio. We need to split the data into training and test data. We use training data to learn patterns in the data and then check if it generalises well on unseen data. Normalisation is done to ensure all the features all weighted appropriately in the training stage. Just because some features have high scale should not translate to having higher influence on the model. Normalisation can be done using Standard Scalar or MinMax Scalar among others. In this particular problem, MinMax Scalar has been used which basically transforms each variable in the range of 0 to 1. Split dataset in train and test as well as into inputs and outputs Normalise the dataset using scalars

```
#splitting the data
# Split Data
Y_train = pd.DataFrame(data = data_train.iloc[:, 1].values, columns = ['target'])
data_train=data_train.drop(["adview"],axis=1)
data_train=data_train.drop(["vidid"],axis=1)
data_train.head()

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data_train, Y_train, test_size=0.2, random_state=42)
print(X_train.shape)

# Normalise Data
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train=scaler.fit_transform(X_train)
X_test=scaler.fit_transform(X_test)

X_test.mean()
```

```
(11688, 7)
0.1802471544523298
```

5. **Model Training:** Different machine learning models can be used for training out of which we can choose whichever gives the best result. We will use the scikitlearn libraries for importing these models and the training them use the fit method and providing necessary labelled data (input and output). We are optimising for mean square error here because it's a regression problem after all. The metrics that we can use for us to compare different model can be mean square error and mean absolute error. Import scikitlearn library Import the model and define Use .fit method with data as arguments to train Calculate errors

Mean Absolute Error (MAE):

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

This calculates the average absolute difference between the true values yi and the predicted values (yi)'. It gives you a sense of how far off the predictions are, on average, in the same units as the target variable.

Mean Squared Error (MSE):

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

This calculates the average squared difference between the true values yi and the predicted values (yi)'. By squaring the differences, larger errors have a more significant impact, which makes this metric sensitive to outliers.

Root Mean Squared Error (RMSE):

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} = \sqrt{\text{MSE}}$$

RMSE is the square root of the Mean Squared Error. It gives you an error metric in the same units as the target variable and penalizes larger errors more than MAE

```
import numpy as np
from sklearn import metrics
from sklearn import linear_model
from sklearn.svm import SVR

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)))

# Convert y_train and y_test to 1-dimensional NumPy arrays
y_train = np.array(y_train).ravel()
y_test = np.array(y_test).ravel()

# Linear Regression
linear_regression = linear_model.LinearRegression()
linear_regression.fit(X_train, y_train)
print_error(X_test, y_test, linear_regression)

# Support Vector Regressor
supportvector_regressor = SVR()
supportvector_regressor.fit(X_train, y_train)
print_error(X_test, y_test, supportvector_regressor)
```

```
Mean Absolute Error: 1638.8610051016046
Mean Squared Error: 67307601.97653732
Root Mean Squared Error: 8204.121036195
Mean Absolute Error: 916.6565552095075
Mean Squared Error: 68656267.86441168
Root Mean Squared Error: 8285.907787588014
```

6. **Using Decision Tree Regressor and Random Forest Regressors.:** Another class of machine learning algorithms include decision trees and random forests. We can import these models from sklearn.tree and then again use the .fit function. We need to give appropriate hyper parameters for them. These are something we can experiment with to achieve better results. Import models Assign hyperparameters for random forest Train the models Calculate errors

```
# Decision Tree Regressor
decision_tree = DecisionTreeRegressor()
decision_tree.fit(X_train, y_train)
print_error(X_test, y_test, decision_tree)

# Random Forest Regressor
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_sampl
random_forest.fit(X_train, y_train)
print_error(X_test, y_test, random_forest)

# Keras Neural Network
model = Sequential()
model.add(Input(shape=(X_train.shape[1],)))
model.add(Dense(64, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(1))

# Output layer
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32)

# Evaluate the model
print_error(X_test, y_test, model)
```

7. **Build an artificial neural network and train it with different layers and hyperparameters.:** The model trains for different epochs (going through dataset once means one epoch) to result in an improved model. We may need to perform hyperparameter tuning (i.e. selecting the best hyperparamters like number of neurons or activation function to yield minimum error).

```
# Artificial Neural Network
import keras
from keras.layers import Dense
ann = keras.models.Sequential([
Dense(6, activation="relu",
input_shape=X_train.shape[1:]),
Dense(6,activation="relu"),
Dense(1)
])
optimizer=keras.optimizers.Adam()
loss=keras.losses.mean_squared_error
ann.compile(optimizer=optimizer,loss=loss,metrics=["mean_squared_error"])
history=ann.fit(X_train,y_train,epochs=100)
 ann.summary()
print_error(X_test,y_test,ann)
```

Layer (type)	Output Shape	Param #
dense_6 (Dense)	(None, 6)	48
dense_7 (Dense)	(None, 6)	42
dense_8 (Dense)	(None, 1)	7

```
Total params: 293 (1.15 KB)
Trainable params: 97 (388.00 B)
Non-trainable params: 0 (0.00 B)
Optimizer params: 196 (788.00 B)
92/92 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
Mean Absolute Error: 1583.663806344592
Mean Squared Error: 67381472.72931203
Root Mean Squared Error: 8208.621853229202
```

8. **Saving the Model:** After choosing the best model here the decidion tree regression model gives the lowest error hence we will save decision tree regression model

```
import joblib
from keras.models import Sequential

# Saving Scikit-Learn models
joblib.dump(decision_tree, "decisiontree_youtubeadview.pkl")

# Saving Keras Artificial Neural Network model
ann.save("ann_youtubeadview.keras")
```

Related work Around the World:

Globally, machine learning and data analytics are extensively used in digital marketing to optimize advertising campaigns, predict user behavior, and personalize content. Companies like Google, Facebook, and Amazon leverage these techniques to analyze massive datasets and improve ad targeting, maximizing ad revenue and user engagement. Predictive analytics in advertising is a rapidly growing field, with continuous research focusing on developing more accurate and efficient models. Advances in neural networks and deep learning further enhance the ability to handle complex data structures and improve prediction capabilities.

Results Obtained:

1. Starting from the given training data set, we imported CSV file, found missing values, and manipulated the categorical data set to have advanced visualization.
2. We used Mean absolute error, mean squared error, and root mean squared error to find predicted values.
3. From sci-kit.learn we import various inbuilt model like linear regression model, and decision tree regression model to find the relationship between attributes.
4. As the resultant decision tree model gave the least error so we chose to save model using joblib libraries

Future Scope of Work:

1. **Feature Engineering:** Exploring additional features, such as video content type, sentiment analysis of comments, and user engagement metrics, could improve prediction accuracy.
2. **Real-Time Prediction:** Developing models capable of real-time ad view prediction to assist advertisers in making instantaneous decisions during ad campaigns.
3. **Scalability:** Optimizing the models to handle larger datasets, potentially integrating data from multiple social media platforms to generalize the prediction models across different video content types and user bases.