Machine Learning Analysis of YouTube Metrics

OUR CONSULTANTS

Aaron Schneberger

Ashok Goyal

Navyasri Pusuluri

Roli Singh

TABLE OF CONTENTS

OBJECTIVES:

- To use unsupervised learning model “K-Means” to predict clusters of YouTube videos.
- To use supervised learning model “Keras” to predict the revenue generated by YouTube videos.
- To use supervised learning model “Keras” to predict the number of views of YouTube videos
- To tune the supervised learning model to find the best hyperparameters.

Introducing the Data

The source data file from Kaggle includes metrics such as:

- Comments added: The number of comments on the video
- Views: The number of views the video has
- Shares: The number of times the video was shared
- Likes: The number of likes the video has
- Dislikes: The number of dislikes the video has
- RPM (USD): The revenue per thousand views the video has
- CPM (USD): The cost per thousand views the video has
- Watch time (hours): The total number of hours the video has been watched.

Data Understanding/Pre-Processing

Data Extraction and Cleaning

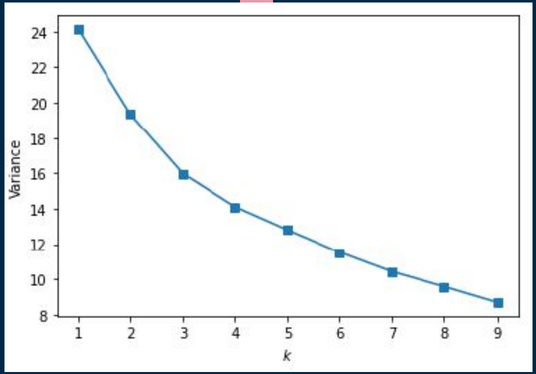
Heat map

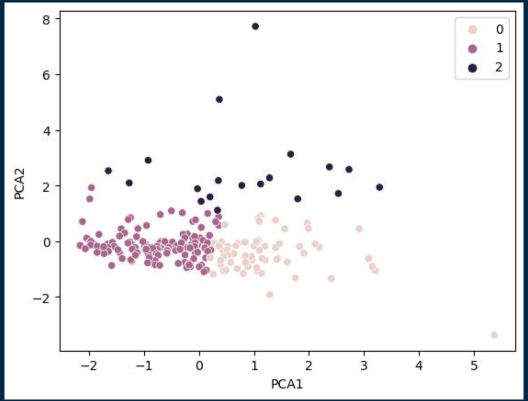
Scaling

PCA

K-Means Clustering

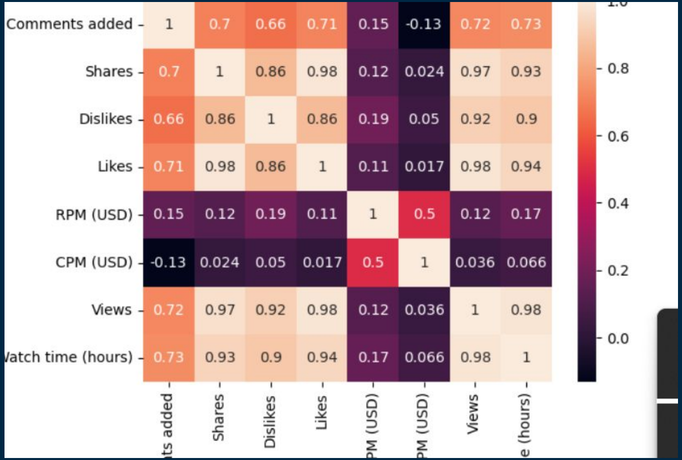
Agglomerative Clustering

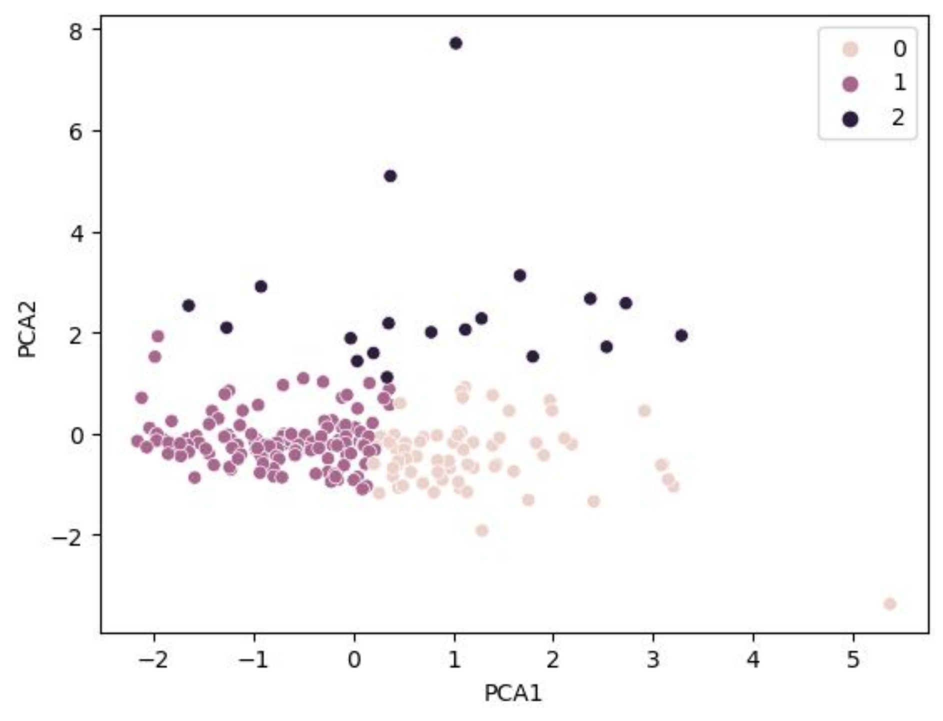




K Means Clustering/ Agglomerative Clustering

- Removed columns from data
- Standardized
- Made a heat map of features
- Used PCA to reduce dimensionality to 4 with a total explained variance of 0.9677





	PC1	PC2	PC3	PC4
Comments added	0.053539	0.615340	-0.203983	0.752476
Shares	0.083922	0.365346	0.307977	-0.272334
Dislikes	0.109409	0.203606	0.018685	-0.199867
Likes	0.078747	0.378704	0.315887	-0.256393
RPM (USD)	0.709812	0.074660	-0.623908	-0.280986
CPM (USD)	0.669248	-0.341088	0.541028	0.371087
Views	0.081488	0.300242	0.222544	-0.171631
Watch time (hours)	0.116416	0.301573	0.178632	-0.088401

DEEP LEARNING REGRESSION

We first tried to use a time series to predict views, but decided the inherent positive trend biased the data

Out[9]:

	Video Length	Video Likes Added	Video Dislikes Added	Video Likes Removed	User Subscriptions Added	User Subscriptions Removed	User Comments Added
0	2191	0	0	0	0	0	0
1	51	0	0	0	1	0	0
2	2686	0	0	0	0	0	0
3	980	0	0	0	0	0	0
4	2904	0	0	0	0	0	0
...	...	...	...	...	...	...	...
111852	311	0	0	0	0	0	0
111853	311	0	0	0	0	0	0
111854	311	0	0	0	0	0	0
111855	311	0	0	0	0	0	0
111856	729	0	0	0	0	0	0

111857 rows × 7 columns

DEEP LEARNING REGRESSION

We changed the revenue per 1000 views to revenue then used it as the target vector:

```
In [9]:  y = df["RPM (USD)"]*df["Views"]/1000

In [11]: X = df.drop(["RPM (USD)", "Views"], axis = 1)
```

DEEP LEARNING REGRESSION

We used the other columns (other than views) as the features:

```
In [11]: X = df.drop(["RPM (USD)", "Views"], axis = 1)

In [12]: X.rename(columns = {"CPM (USD)": "Cost"}, inplace = True)

In [13]: X
Out[13]:
```

	Comments added	Shares	Dislikes	Likes	Cost	Watch time (hours)
1	907	9583	942	46903	16089.429765	65850.7042
2	412	4	4	130	14.339369	200.2966
3	402	152	15	881	249.688250	3687.3387
4	375	367	22	2622	393.686852	2148.3110
5	329	118	15	590	99.710325	1034.3945
...	...	...	...	...	...	...
218	4	5	0	30	46.287850	9.6188
219	3	5	1	48	15.874896	56.5930
220	3	0	0	44	8.546608	19.2752
221	3	1	0	35	9.077390	22.5450
222	2	5	0	38	13.353364	57.6363

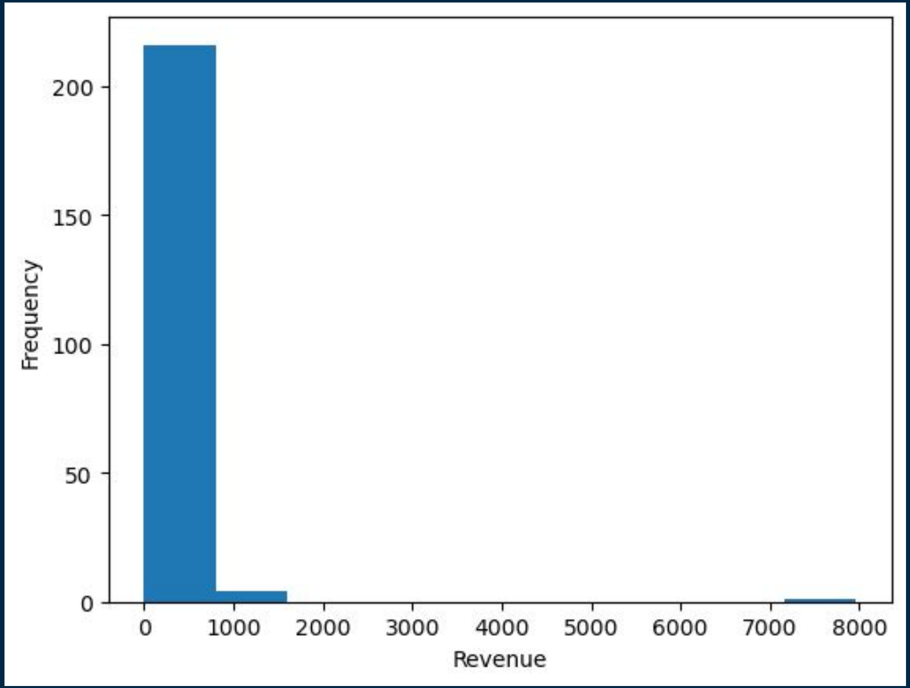
FIRST ATTEMPT AT REGRESSION

```
In [89]: # Define the model with six hidden layers and an output layer with one unit
         nn = Sequential()

         # First hidden layer
         nn.add(Dense(units=8, activation="relu", input_dim=6))

         for i in range(5):
             nn.add(Dense(units=8, activation="relu"))

         # Output layer
         nn.add(Dense(units=1, activation='relu'))

         # Check the structure of the model
         nn.summary()
```

RESULTS OF FIRST ATTEMPT

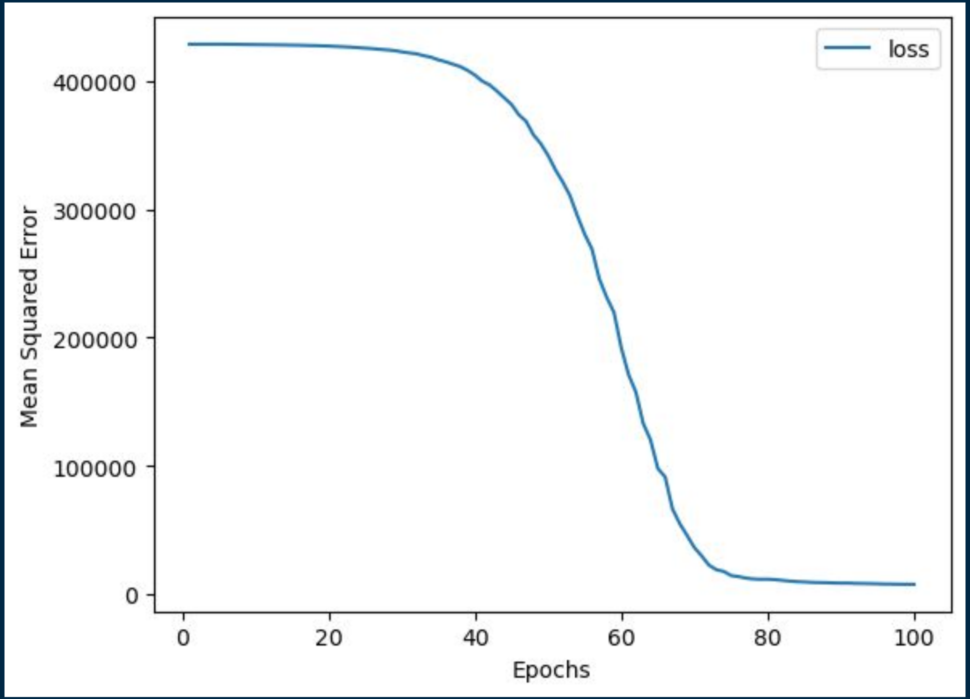
- Test data had mean squared error of 7036.71
- Root mean error was 83.89
- The mean of the revenue column is $132.21, and the max is $7963.86

DISTRIBUTION OF REVENUE



Root mean error of 83.89 quite high for this distribution

LOSS VERSUS EPOCHS



We had some overfitting in our first attempt

LOSS FUNCTION

- Decided that absolute percentage error was a better loss function

```
In [92]: # Evaluate the model using the test data
         model_loss, model_percent_error = nn.evaluate(X_test_scaled,y_test,verbose=2)
         print(f"Loss: {model_loss}, Mean Absolute Percent Error: {model_percent_error}")

         2/2 - 0s - loss: 7036.7100 - mean_absolute_percentage_error: 424.5189 - 107ms/epoch - 54ms/step
         Loss: 7036.7099609375, Mean Absolute Percent Error: 424.5188903808594
```

KERAS TUNER

- Used keras tuner to find better hyperparameters

```
In [99]: # Create a method that creates a new Sequential model with hyperparameter options
def create_model_percent_error(hp):
    nn_model = Sequential()

    # Allow kerastuner to decide which activation function to use in hidden layers
    activation = hp.Choice('activation',['relu','tanh'])

    # Allow kerastuner to decide number of neurons in first layer
    nn_model.add(Dense(units=hp.Int('first_units',
        min_value=1,
        max_value=15,
        step=5), activation=activation, input_dim=6))

    # Allow kerastuner to decide number of hidden layers and neurons in hidden layers
    for i in range(hp.Int('num_layers', 1, 7)):
        nn_model.add(Dense(units=hp.Int('units_' + str(i),
            min_value=1,
            max_value=15,
            step=5),
            activation=activation))

    nn_model.add(Dense(units=1, activation="relu"))

    # Compile the model
    nn_model.compile(loss="mean_absolute_percentage_error", optimizer='adam',metrics = ['mean_absolute_percentage_error

    return nn_model
```

RESULTS OF TUNING

Here are hyperparameters for best model

In [107]:
```
# Find top model hyperparameters and print the values for mean absolute percentage error loss function
top_hyper_percent_loss = tuner.get_best_hyperparameters(1)
for param in top_hyper_percent_loss:
    print(param.values)
```

```
{'activation': 'relu', 'first_units': 6, 'num_layers': 4, 'units_0': 11, 'units_1': 11, 'units_2': 1, 'units_3': 11,
'units_4': 11, 'units_5': 6, 'units_6': 1, 'tuner/epochs': 200, 'tuner/initial_epoch': 67, 'tuner/bracket': 2, 'tune
r/round': 2, 'tuner/trial_id': '0221'}
```

RESULTS OF TUNING

Best model had mean absolute percentage error of 37.86 during validation

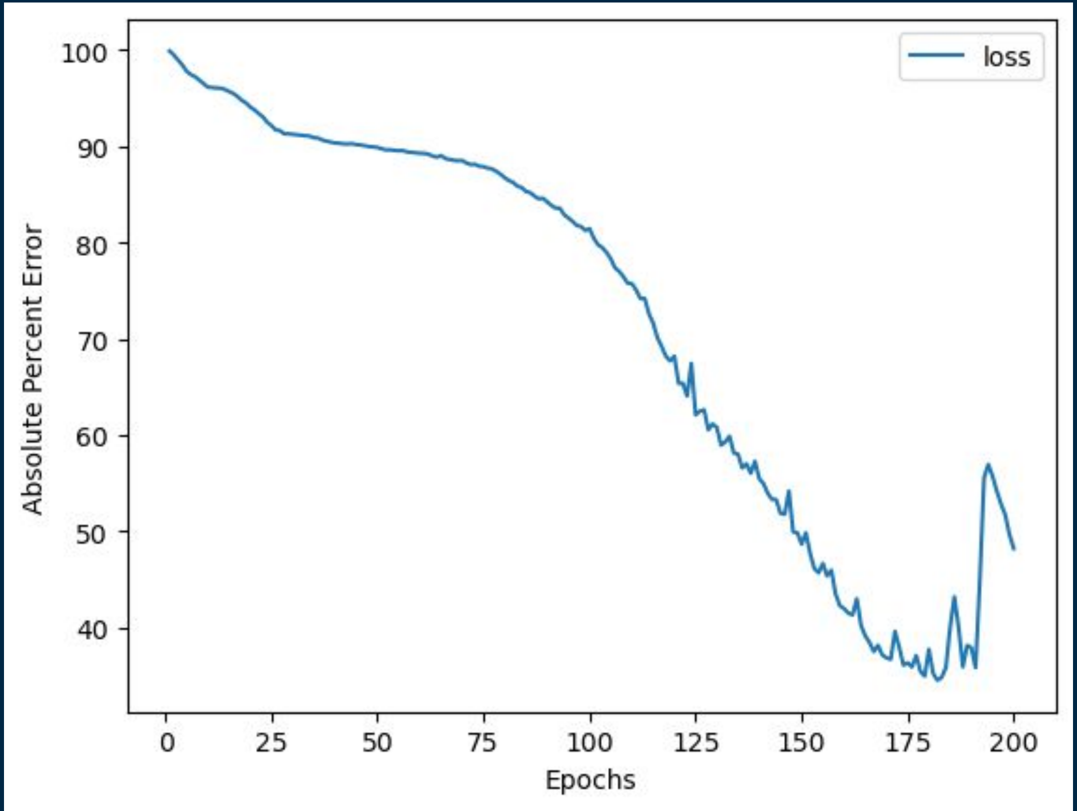
```
# Find the best model for mean absolute percentage error
top_model_mean_absolute_percentage_error = tuner.get_best_models(1)[0]

model_loss = top_model_mean_absolute_percentage_error.evaluate(X_test_scaled,y_test,verbose=1)
print(f"Mean Absolute Percentage Error: {model_loss}")
```

```
2/2 [==============================] - 0s 4ms/step - loss: 37.8564 - mean_absolute_percentage_error: 37.8564
Mean Absolute Percentage Error: [37.856414794921875, 37.856414794921875]
```

But we felt there was overfitting

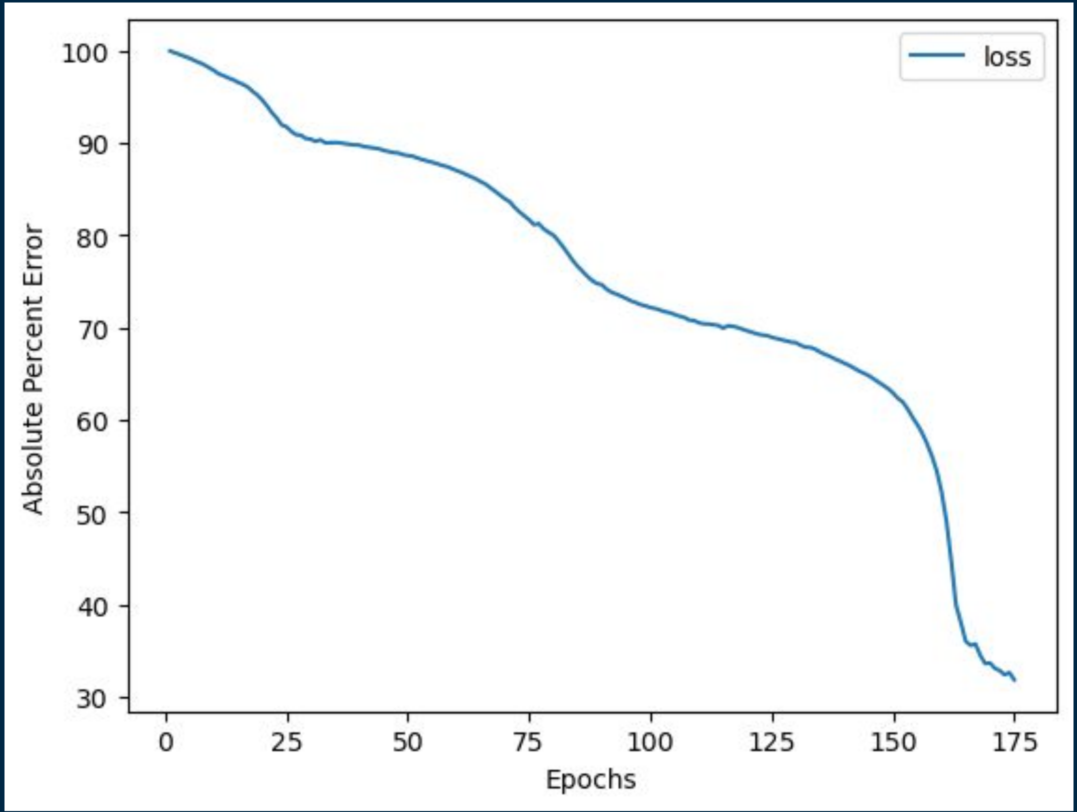
RESULTS OF TUNING

We ran the training again and saw the loss graph to the right



RESULTS OF TUNING

After using 175 for our number of epochs, we got a better loss curve:



RESULTS OF TUNING

When evaluating the model, we got a mean error of 32.74%

In [38]:
```
# Evaluate the model using the test data
model_loss = nn_final.evaluate(X_test_scaled,y_test,verbose=1)
print(f"Mean Absolute Percent Error: {model_loss}")
```

```
2/2 [==============================] - 0s 3ms/step - loss: 32.7416 - mean_absolute_percentage_error: 32.7416
Mean Absolute Percent Error: [32.741600036621094, 32.741600036621094]
```

SUMMARY

- Clustered videos with k-means and agglomerative
- Created deep learning networks to predict revenue
- Achieved a mean percent error of 32.74%

THANKS