TikTok Project

Course 5 - Regression Analysis: Simplify complex data relationships

You are a data professional at TikTok. The data team is working towards building a machine learning model that can be used to determine whether a video contains a claim or whether it offers an opinion. With a successful prediction model, TikTok can reduce the backlog of user reports and prioritize them more efficiently.

The team is getting closer to completing the project, having completed an initial plan of action, initial Python coding work, EDA, and hypothesis testing.

The TikTok team has reviewed the results of the hypothesis testing. TikTok's Operations Lead, Maika Abadi, is interested in how different variables are associated with whether a user is verified. Earlier, the data team observed that if a user is verified, they are much more likely to post opinions. Now, the data team has decided to explore how to predict verified status to help them understand how video characteristics relate to verified users. Therefore, you have been asked to conduct a logistic regression using verified status as the outcome variable. The results may be used to inform the final model related to predicting whether a video is a claim vs an opinion.

A notebook was structured and prepared to help you in this project. Please complete the following questions.

Course 5 End-of-course project: Regression modeling

In this activity, you will build a logistic regression model in Python. As you have learned, logistic regression helps you estimate the probability of an outcome. For data science professionals, this is a useful skill because it allows you to consider more than one variable against the variable you're measuring against. This opens the door for much more thorough and flexible analysis to be completed.

The purpose of this project is to demostrate knowledge of EDA and regression models.

The goal is to build a logistic regression model and evaluate the model.

This activity has three parts:
Part 1: EDA & Checking Model Assumptions

- What are some purposes of EDA before constructing a logistic regression model?

Part 2: Model Building and Evaluation

- What resources do you find yourself using as you complete this stage?

Part 3: Interpreting Model Results

- What key insights emerged from your model(s)?
- What business recommendations do you propose based on the models built?

Follow the instructions and answer the question below to complete the activity. Then, you will complete an executive summary using the questions listed on the PACE Strategy Document.

Be sure to complete this activity before moving on. The next course item will provide you with a completed exemplar to compare to your own work.

Build a regression model

Task 1. Imports and loading

Import the data and packages that you've learned are needed for building regression models.

In [1]:
```
# Import packages for data manipulation
import pandas as pd
import numpy as np

# Import packages for data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Import packages for data preprocessing
from sklearn.preprocessing import OneHotEncoder
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.utils import resample

# Import packages for data modeling
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

Load the TikTok dataset.

Note: As shown in this cell, the dataset has been automatically loaded in for you. You do not need to download the .csv file, or provide more code, in order to access the dataset and proceed with this lab. Please continue with this activity by completing the following instructions.

In [2]:
```
# Load dataset into dataframe
data = pd.read_csv("tiktok_dataset.csv")
```

Response:

The purposes of EDA before constructing a logistic regression model are

1. to identify data anomalies such as outliers and class imbalance that might affect the modeling;
2. to verify model assumptions such as no severe multicollinearity.

Task 2a. Explore data with EDA

Analyze the data and check for and handle missing values and duplicates.

Inspect the first five rows of the dataframe.

In [3]:
```
# Display first few rows
data.head()
```

Out[3]:

	#	claim_status	video_id	video_duration_sec	video_transcription_text	verifiec
0	1	claim	7017666017	59	someone shared with me that drone deliveries a...	no
1	2	claim	4014381136	32	someone shared with me that there are more mic...	no
2	3	claim	9859838091	31	someone shared with me that american industria...	no
3	4	claim	1866847991	25	someone shared with me that the metro of st. p...	no
4	5	claim	7105231098	19	someone shared with me that the number of busi...	no

Get the number of rows and columns in the dataset.

In [4]:
```
# Get number of rows and columns
data.shape
```

Out[4]: (19382, 12)

Get the data types of the columns.

In [5]:
```python
# Get data types of columns
data.dtypes
```

Out[5]:
```
#                              int64
claim_status                  object
video_id                       int64
video_duration_sec             int64
video_transcription_text      object
verified_status               object
author_ban_status             object
video_view_count             float64
video_like_count             float64
video_share_count            float64
video_download_count         float64
video_comment_count          float64
dtype: object
```

Get basic information about the dataset.

In [6]:
```python
# Get basic information
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19382 entries, 0 to 19381
Data columns (total 12 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   #                         19382 non-null  int64  
 1   claim_status              19084 non-null  object 
 2   video_id                  19382 non-null  int64  
 3   video_duration_sec        19382 non-null  int64  
 4   video_transcription_text  19084 non-null  object 
 5   verified_status           19382 non-null  object 
 6   author_ban_status         19382 non-null  object 
 7   video_view_count          19084 non-null  float64
 8   video_like_count          19084 non-null  float64
 9   video_share_count         19084 non-null  float64
 10  video_download_count      19084 non-null  float64
 11  video_comment_count       19084 non-null  float64
dtypes: float64(5), int64(3), object(4)
memory usage: 1.8+ MB
```

Generate basic descriptive statistics about the dataset.

In [7]:
```python
# Generate basic descriptive stats
data.describe()
```

Out[7]:

	#	video_id	video_duration_sec	video_view_count	video_like
count	19382.000000	1.938200e+04	19382.000000	19084.000000	19084.(
mean	9691.500000	5.627454e+09	32.421732	254708.558688	84304.(
std	5595.245794	2.536440e+09	16.229967	322893.280814	133420.
min	1.000000	1.234959e+09	5.000000	20.000000	0.(
25%	4846.250000	3.430417e+09	18.000000	4942.500000	810.:
50%	9691.500000	5.618664e+09	32.000000	9954.500000	3403.!
75%	14536.750000	7.843960e+09	47.000000	504327.000000	125020.(
max	19382.000000	9.999873e+09	60.000000	999817.000000	657830.(

Check for and handle missing values.

In [8]:
```
# Check for missing values
data.isna().sum()
```

Out[8]:
```
#                            0
claim_status               298
video_id                     0
video_duration_sec           0
video_transcription_text   298
verified_status              0
author_ban_status            0
video_view_count           298
video_like_count           298
video_share_count          298
video_download_count       298
video_comment_count        298
dtype: int64
```

In [9]:
```
# Drop rows with missing values
data = data.dropna(axis=0)
```

In [10]:
```
# Display first few rows after handling missing values
data.head()
```

Out[10]:

	#	claim_status	video_id	video_duration_sec	video_transcription_text	verifiec
0	1	claim	7017666017	59	someone shared with me that drone deliveries a...	no
1	2	claim	4014381136	32	someone shared with me that there are more mic...	no
2	3	claim	9859838091	31	someone shared with me that american industria...	no
3	4	claim	1866847991	25	someone shared with me that the metro of st. p...	no
4	5	claim	7105231098	19	someone shared with me that the number of busi...	no

Check for and handle duplicates.

In [11]:

```
# Check for duplicates
data.duplicated().sum()
```

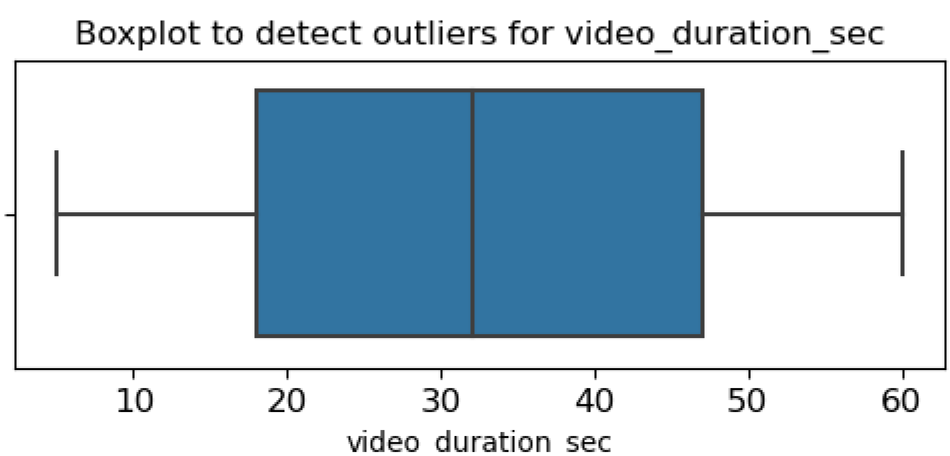
Out[11]: 0

Note: There does not seem to be any duplicates.

Check for and handle outliers.

In [12]:

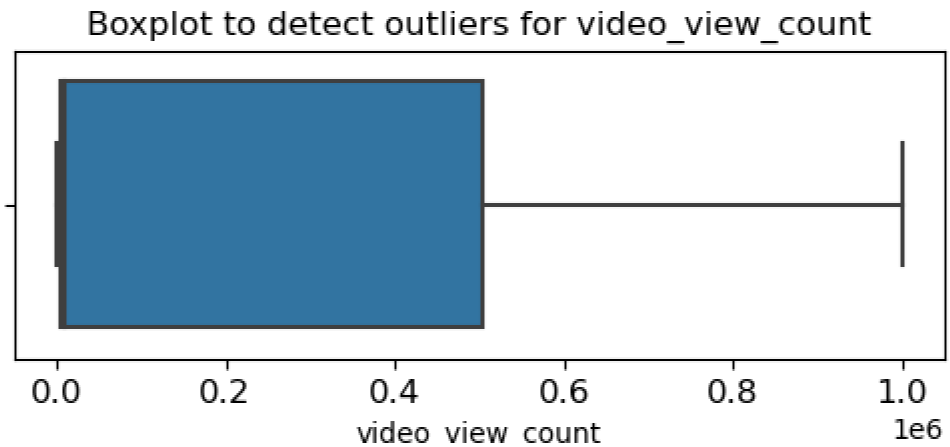
```
# Create a boxplot to visualize distribution of `video_duration_sec`
plt.figure(figsize=(6,2))
plt.title('Boxplot to detect outliers for video_duration_sec', fontsize=12)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
sns.boxplot(x=data['video_duration_sec'])
plt.show()
```



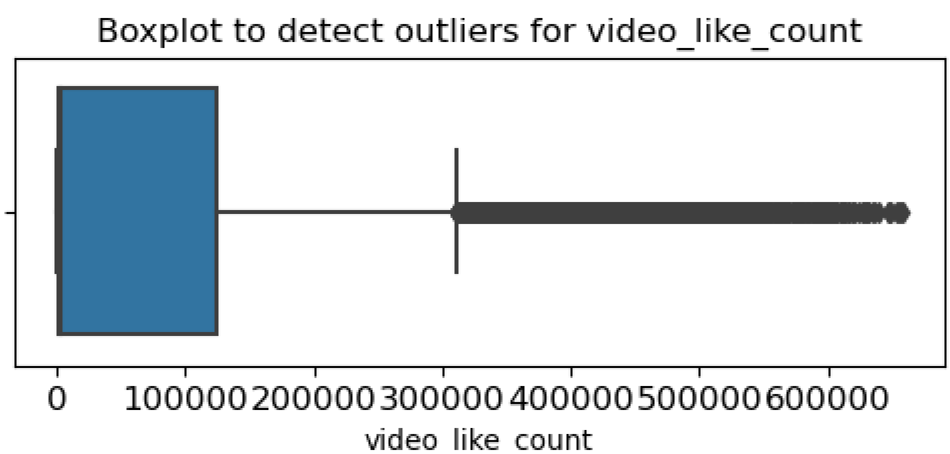
In [13]:

```
# Create a boxplot to visualize distribution of `video_view_count`
plt.figure(figsize=(6,2))
```

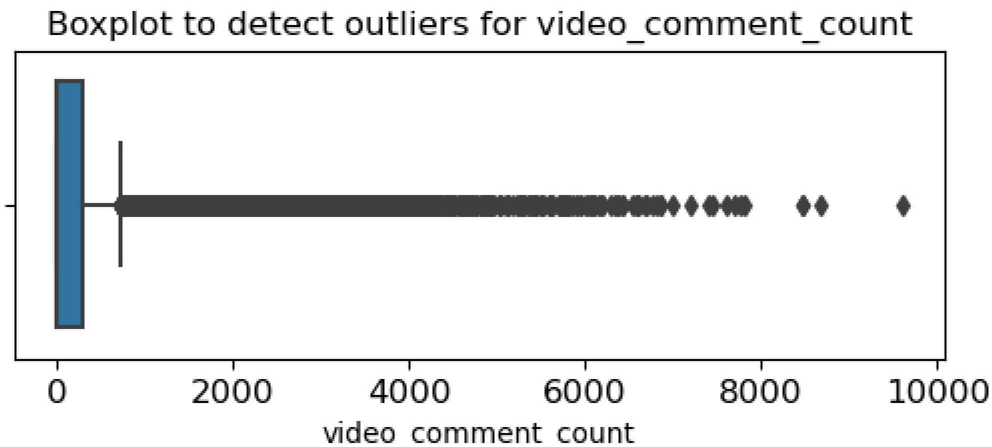
```
plt.title('Boxplot to detect outliers for video_view_count', fontsize=12)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
sns.boxplot(x=data['video_view_count'])
plt.show()
```



In [14]:
```
# Create a boxplot to visualize distribution of `video_like_count`
plt.figure(figsize=(6,2))
plt.title('Boxplot to detect outliers for video_like_count', fontsize=12)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
sns.boxplot(x=data['video_like_count'])
plt.show()
```



In [15]:
```
# Create a boxplot to visualize distribution of `video_comment_count`
plt.figure(figsize=(6,2))
plt.title('Boxplot to detect outliers for video_comment_count', fontsize=12)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
sns.boxplot(x=data['video_comment_count'])
plt.show()
```



In [16]:
```
# Check for and handle outliers

percentile25 = data["video_like_count"].quantile(0.25)
percentile75 = data["video_like_count"].quantile(0.75)

iqr = percentile75 - percentile25
upper_limit = percentile75 + 1.5 * iqr

data.loc[data["video_like_count"] > upper_limit, "video_like_count"] = upper
```

In [17]:
```
# Check for and handle outliers

percentile25 = data["video_comment_count"].quantile(0.25)
percentile75 = data["video_comment_count"].quantile(0.75)

iqr = percentile75 - percentile25
upper_limit = percentile75 + 1.5 * iqr

data.loc[data["video_comment_count"] > upper_limit, "video_comment_count"] =
```

Check class balance.

In [18]:
```
# Check class balance
data["verified_status"].value_counts(normalize=True)
```

Out[18]:
```
verified_status
not verified    0.93712
verified        0.06288
Name: proportion, dtype: float64
```

Approximately 94.2% of the dataset represents videos posted by unverified accounts and 5.8% represents videos posted by verified accounts. So the outcome variable is not very balanced.

Use resampling to create class balance in the outcome variable, if needed.

In [19]:
```
# Use resampling to create class balance in the outcome variable, if needed

# Identify data points from majority and minority classes
```

```
data_majority = data[data["verified_status"] == "not verified"]
data_minority = data[data["verified_status"] == "verified"]

# Upsample the minority class (which is "verified")
data_minority_upsampled = resample(data_minority,
                                   replace=True,                 # to sample w
                                   n_samples=len(data_majority), # to match ma
                                   random_state=0)               # to create r

# Combine majority class with upsampled minority class
data_upsampled = pd.concat([data_majority, data_minority_upsampled]).reset_i

# Display new class counts
data_upsampled["verified_status"].value_counts()
```

Out[19]:

```
verified_status
not verified    17884
verified        17884
Name: count, dtype: int64
```

Get the average `video_transcription_text` length for videos posted by verified accounts and the average `video_transcription_text` length for videos posted by unverified accounts.

In [20]:

```
# Get the average `video_transcription_text` length for claims and the avera
data_upsampled[["verified_status", "video_transcription_text"]].groupby(by="
```

Out[20]:

verified_status	video_transcription_text
not verified	89.401141
verified	84.569559

Extract the length of each `video_transcription_text` and add this as a column to the dataframe, so that it can be used as a potential feature in the model.

In [21]:

```
# Extract the length of each `video_transcription_text` and add this as a co
data_upsampled["text_length"] = data_upsampled["video_transcription_text"].a
```

In [22]:

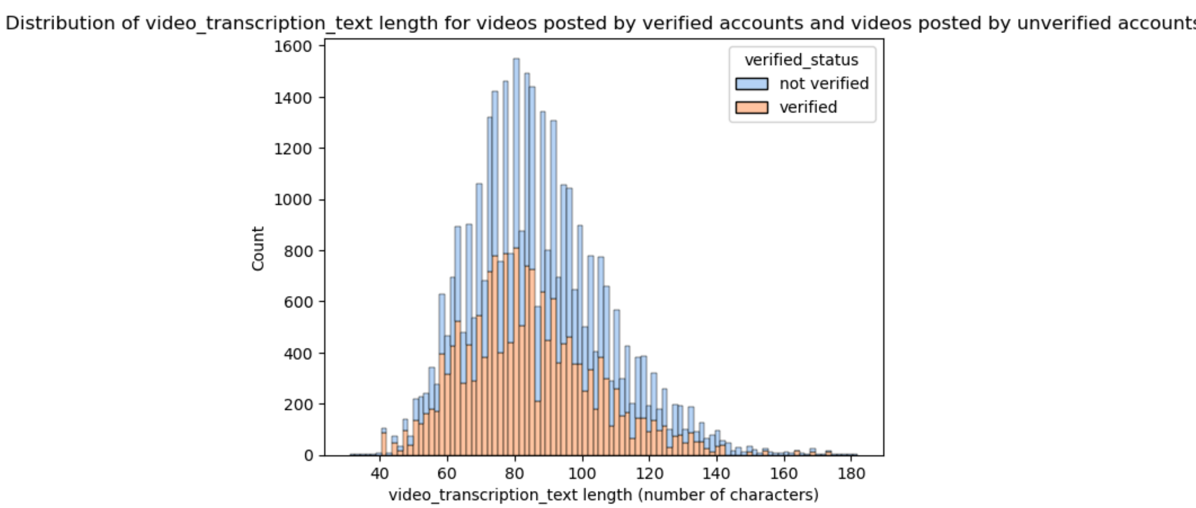
```
# Display first few rows of dataframe after adding new column
data_upsampled.head()
```

Out[22]:

	#	claim_status	video_id	video_duration_sec	video_transcription_text	verified
0	1	claim	7017666017	59	someone shared with me that drone deliveries a...	no
1	2	claim	4014381136	32	someone shared with me that there are more mic...	no
2	3	claim	9859838091	31	someone shared with me that american industria...	no
3	4	claim	1866847991	25	someone shared with me that the metro of st. p...	no
4	5	claim	7105231098	19	someone shared with me that the number of busi...	no

Visualize the distribution of `video_transcription_text` length for videos posted by verified accounts and videos posted by unverified accounts.

In [23]:
```python
# Visualize the distribution of `video_transcription_text` length for videos
# Create two histograms in one plot
sns.histplot(data=data_upsampled, stat="count", multiple="stack", x="text_le
             hue="verified_status", element="bars", legend=True)
plt.title("Seaborn Stacked Histogram")
plt.xlabel("video_transcription_text length (number of characters)")
plt.ylabel("Count")
plt.title("Distribution of video_transcription_text length for videos postec
plt.show()
```



Task 2b. Examine correlations

Next, code a correlation matrix to help determine most correlated variables.

In [24]:
```python
# Code a correlation matrix to help determine most correlated variables
data_upsampled.corr(numeric_only=True)
```

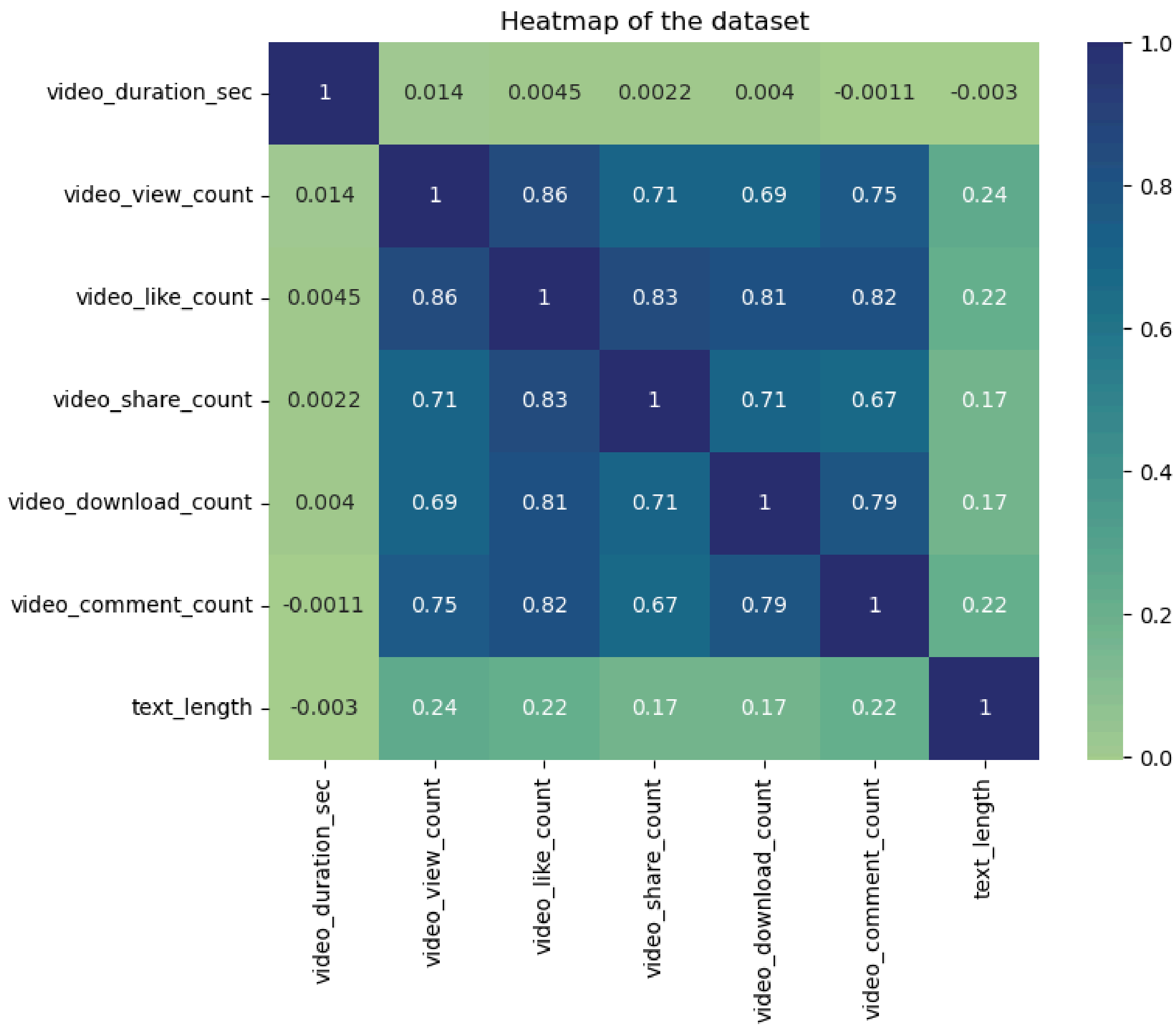
Out[24]:

	#	video_id	video_duration_sec	video_view_count
#	1.000000	-0.000853	-0.011729	-0.697007
video_id	-0.000853	1.000000	0.011859	0.002554
video_duration_sec	-0.011729	0.011859	1.000000	0.013589
video_view_count	-0.697007	0.002554	0.013589	1.000000
video_like_count	-0.626385	0.005993	0.004494	0.856937
video_share_count	-0.504015	0.010515	0.002206	0.711313
video_download_count	-0.487096	0.008753	0.003989	0.690048
video_comment_count	-0.608773	0.012674	-0.001086	0.748361
text_length	-0.193677	-0.007083	-0.002981	0.244693

Visualize a correlation heatmap of the data.

In [26]:

```
# Create a heatmap to visualize how correlated variables are
plt.figure(figsize=(8, 6))
sns.heatmap(
    data_upsampled[["video_duration_sec", "claim_status", "author_ban_status
                    "video_like_count", "video_share_count", "video_download
    .corr(numeric_only=True),
    annot=True,
    cmap="crest")
plt.title("Heatmap of the dataset")
plt.show()
```



One of the model assumptions for logistic regression is no severe multicollinearity among the features. Take this into consideration as you examine the heatmap and choose which features to proceed with.

Response: The above heatmap shows that the following pair of variables are strongly correlated: `video_view_count` and `video_like_count` (0.86 correlation coefficient).

One of the model assumptions for logistic regression is no severe multicollinearity among the features. To build a logistic regression model that meets this assumption, you could exclude `video_like_count`. And among the variables that quantify video metrics, you could keep `video_view_count`, `video_share_count`, `video_download_count`, and `video_comment_count` as features.

Task 3a. Select variables

Set your Y and X variables.

Select the outcome variable.

In [27]:
```
# Select outcome variable
y = data_upsampled["verified_status"]
```

Select the features.

In [28]:
```
# Select features
X = data_upsampled[["video_duration_sec", "claim_status", "author_ban_status

# Display first few rows of features dataframe
X.head()
```

Out[28]:

	video_duration_sec	claim_status	author_ban_status	video_view_count	video_sha
0	59	claim	under review	343296.0	
1	32	claim	active	140877.0	
2	31	claim	active	902185.0	
3	25	claim	active	437506.0	
4	19	claim	active	56167.0	

Note: The `#` and `video_id` columns are not selected as features here, because they do not seem to be helpful for predicting whether a video presents a claim or an opinion. Also, `video_like_count` is not selected as a feature here, because it is strongly correlated with other features, as discussed earlier. And logistic regression has a no multicollinearity model assumption that needs to be met.

Task 3b. Train-test split

Split the data into training and testing sets.

In [29]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, ra
```

Confirm that the dimensions of the training and testing sets are in alignment.

In [30]:
```
# Get shape of each training and testing set
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[30]:
```
((26826, 7), (8942, 7), (26826,), (8942,))
```

Notes:

- The number of features (`7`) aligns between the training and testing sets.

- The number of rows aligns between the features and the outcome variable for training (`26826`) and testing (`8942`).

Task 3c. Encode variables

Check the data types of the features.

In [31]:
```
# Check data types
X_train.dtypes
```

Out[31]:
```
video_duration_sec        int64
claim_status             object
author_ban_status        object
video_view_count        float64
video_share_count       float64
video_download_count    float64
video_comment_count     float64
dtype: object
```

In [32]:
```
# Get unique values in `claim_status`
X_train["claim_status"].unique()
```

Out[32]:
```
array(['opinion', 'claim'], dtype=object)
```

In [33]:
```
# Get unique values in `author_ban_status`
X_train["author_ban_status"].unique()
```

Out[33]:
```
array(['active', 'under review', 'banned'], dtype=object)
```

As shown above, the `claim_status` and `author_ban_status` features are each of data type `object` currently. In order to work with the implementations of models through `sklearn`, these categorical features will need to be made numeric. One way to do this is through one-hot encoding.

Encode categorical features in the training set using an appropriate method.

In [34]:
```
# Select the training features that needs to be encoded
X_train_to_encode = X_train[["claim_status", "author_ban_status"]]

# Display first few rows
X_train_to_encode.head()
```

Out[34]:

	claim_status	author_ban_status
33058	opinion	active
20491	opinion	active
25583	opinion	active
18474	opinion	active
27312	opinion	active

In [62]:
```
# Set up an encoder for one-hot encoding the categorical features
X_encoder = OneHotEncoder(drop='first', sparse_output=False)
```

In [63]:
```
# Fit and transform the training features using the encoder
X_train_encoded = X_encoder.fit_transform(X_train_to_encode)
```

In [64]:
```
# Get feature names from encoder
X_encoder.get_feature_names_out()
```

Out[64]:
```
array(['claim_status_opinion', 'author_ban_status_banned',
       'author_ban_status_under review'], dtype=object)
```

In [38]:
```
# Display first few rows of encoded training features
X_train_encoded
```

Out[38]:
```
array([[1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       ...,
       [1., 0., 0.],
       [1., 0., 0.],
       [0., 1., 0.]])
```

In [39]:
```
# Place encoded training features (which is currently an array) into a dataf
X_train_encoded_df = pd.DataFrame(data=X_train_encoded, columns=X_encoder.ge

# Display first few rows
X_train_encoded_df.head()
```

Out[39]:

	claim_status_opinion	author_ban_status_banned	author_ban_status_under review
0	1.0	0.0	0.0
1	1.0	0.0	0.0
2	1.0	0.0	0.0
3	1.0	0.0	0.0
4	1.0	0.0	0.0

In [40]:
```
# Display first few rows of `X_train` with `claim_status` and `author_ban_st
X_train.drop(columns=["claim_status", "author_ban_status"]).head()
```

Out[40]:

	video_duration_sec	video_view_count	video_share_count	video_download_co
33058	33	2252.0	23.0	
20491	52	6664.0	550.0	5
25583	37	6327.0	257.0	
18474	57	1702.0	28.0	
27312	21	3842.0	101.0	

In [41]:

```
# Concatenate `X_train` and `X_train_encoded_df` to form the final dataframe
# Note: Using `.reset_index(drop=True)` to reset the index in X_train after
# so that the indices align with those in `X_train_encoded_df` and `count_df
X_train_final = pd.concat([X_train.drop(columns=["claim_status", "author_bar

# Display first few rows
X_train_final.head()
```

Out[41]:

	video_duration_sec	video_view_count	video_share_count	video_download_count
0	33	2252.0	23.0	4.0
1	52	6664.0	550.0	53.0
2	37	6327.0	257.0	3.0
3	57	1702.0	28.0	0.0
4	21	3842.0	101.0	1.0

Check the data type of the outcome variable.

In [42]:

```
# Check data type of outcome variable
y_train.dtype
```

Out[42]:

```
dtype('O')
```

In [43]:

```
# Get unique values of outcome variable
y_train.unique()
```

Out[43]:

```
array(['verified', 'not verified'], dtype=object)
```

A shown above, the outcome variable is of data type `object` currently. One-hot encoding can be used to make this variable numeric.

Encode categorical values of the outcome variable the training set using an appropriate method.

In [44]:
```python
# Set up an encoder for one-hot encoding the categorical outcome variable
y_encoder = OneHotEncoder(drop='first', sparse_output=False)
```

In [45]:
```python
# Encode the training outcome variable
# Notes:
#   - Adjusting the shape of `y_train` before passing into `.fit_transform()
#   - Using `.ravel()` to flatten the array returned by `.fit_transform()`,
y_train_final = y_encoder.fit_transform(y_train.values.reshape(-1, 1)).ravel

# Display the encoded training outcome variable
y_train_final
```

```
/opt/conda/lib/python3.11/site-packages/sklearn/preprocessing/_encoders.py:9
72: FutureWarning: `sparse` was renamed to `sparse_output` in version 1.2 an
d will be removed in 1.4. `sparse_output` is ignored unless you leave `spars
e` to its default value.
  warnings.warn(
```

Out[45]:
```
array([1., 1., 1., ..., 1., 1., 0.])
```

Task 3d. Model building

Construct a model and fit it to the training set.

In [46]:
```python
# Construct a logistic regression model and fit it to the training set
log_clf = LogisticRegression(random_state=0, max_iter=800).fit(X_train_final
```

Task 4a. Results and evaluation

Evaluate your model.

Encode categorical features in the testing set using an appropriate method.

In [47]:
```python
# Select the testing features that needs to be encoded
X_test_to_encode = X_test[["claim_status", "author_ban_status"]]

# Display first few rows
X_test_to_encode.head()
```

Out[47]:

	claim_status	author_ban_status
21061	opinion	active
31748	opinion	active
20197	claim	active
5727	claim	active
11607	opinion	active

In [48]:
```python
# Transform the testing features using the encoder
X_test_encoded = X_encoder.transform(X_test_to_encode)
```

```
# Display first few rows of encoded testing features
X_test_encoded
```

Out[48]:

```
array([[1., 0., 0.],
       [1., 0., 0.],
       [0., 0., 0.],
       ...,
       [1., 0., 0.],
       [0., 0., 1.],
       [1., 0., 0.]])
```

In [49]:

```
# Place encoded testing features (which is currently an array) into a datafr
X_test_encoded_df = pd.DataFrame(data=X_test_encoded, columns=X_encoder.get_

# Display first few rows
X_test_encoded_df.head()
```

Out[49]:

	claim_status_opinion	author_ban_status_banned	author_ban_status_under review
0	1.0	0.0	0.0
1	1.0	0.0	0.0
2	0.0	0.0	0.0
3	0.0	0.0	0.0
4	1.0	0.0	0.0

In [50]:

```
# Display first few rows of `X_test` with `claim_status` and `author_ban_sta
X_test.drop(columns=["claim_status", "author_ban_status"]).head()
```

Out[50]:

	video_duration_sec	video_view_count	video_share_count	video_download_cou
21061	41	2118.0	57.0	
31748	27	5701.0	157.0	
20197	31	449767.0	75385.0	595
5727	19	792813.0	56597.0	514
11607	54	2044.0	68.0	1

In [51]:

```
# Concatenate `X_test` and `X_test_encoded_df` to form the final dataframe f
# Note: Using `.reset_index(drop=True)` to reset the index in X_test after c
# so that the indices align with those in `X_test_encoded_df` and `test_cour
X_test_final = pd.concat([X_test.drop(columns=["claim_status", "author_ban_s

# Display first few rows
X_test_final.head()
```

Out[51]:

	video_duration_sec	video_view_count	video_share_count	video_download_count
0	41	2118.0	57.0	5.0
1	27	5701.0	157.0	1.0
2	31	449767.0	75385.0	5956.0
3	19	792813.0	56597.0	5146.0
4	54	2044.0	68.0	19.0

Test the logistic regression model. Use the model to make predictions on the encoded testing set.

In [52]:
```
# Use the logistic regression model to get predictions on the encoded testir
y_pred = log_clf.predict(X_test_final)
```

Display the predictions on the encoded testing set.

In [53]:
```
# Display the predictions on the encoded testing set
y_pred
```

Out[53]:
```
array([1., 1., 0., ..., 1., 0., 1.])
```

Display the true labels of the testing set.

In [54]:
```
# Display the true labels of the testing set
y_test
```

Out[54]:
```
21061        verified
31748        verified
20197        verified
5727     not verified
11607    not verified
             ...
14756    not verified
26564        verified
14800    not verified
35705        verified
31060        verified
Name: verified_status, Length: 8942, dtype: object
```

Encode the true labels of the testing set so it can be compared to the predictions.

In [55]:
```
# Encode the testing outcome variable
# Notes:
#   - Adjusting the shape of `y_test` before passing into `.transform()`, si
#   - Using `.ravel()` to flatten the array returned by `.transform()`, so t
y_test_final = y_encoder.transform(y_test.values.reshape(-1, 1)).ravel()
```

```
# Display the encoded testing outcome variable
y_test_final
```

Out[55]:

```
array([1., 1., 1., ..., 0., 1., 1.])
```

Confirm again that the dimensions of the training and testing sets are in alignment since additional features were added.

In [56]:

```
# Get shape of each training and testing set
X_train_final.shape, y_train_final.shape, X_test_final.shape, y_test_final.s
```

Out[56]:

```
((26826, 8), (26826,), (8942, 8), (8942,))
```

Note:

- The number of features (`8`) aligns between the training and testing sets.
- The number of rows aligns between the features and the outcome variable for training (`26826`) and testing (`8942`).

Task 4b. Visualize model results

Create a confusion matrix to visualize the results of the logistic regression model.

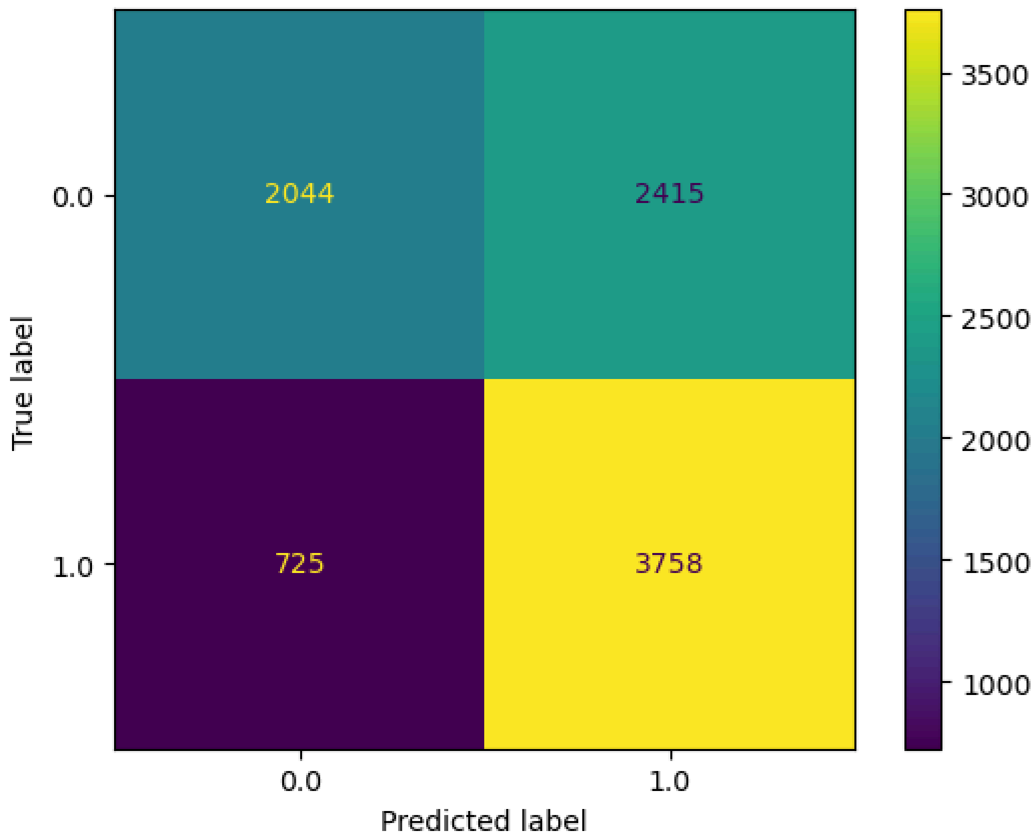
In [57]:

```
# Compute values for confusion matrix
log_cm = confusion_matrix(y_test_final, y_pred, labels=log_clf.classes_)

# Create display of confusion matrix
log_disp = ConfusionMatrixDisplay(confusion_matrix=log_cm, display_labels=lo

# Plot confusion matrix
log_disp.plot()

# Display plot
plt.show()
```



In [65]:
```
(3758+2044) / (3758 + 725 + 2044 + 2415)
```

Out[65]:
```
0.6488481324088571
```

Notes:

The upper-left quadrant displays the number of true negatives: the number of videos posted by unverified accounts that the model accurately classified as so.

The upper-right quadrant displays the number of false positives: the number of videos posted by unverified accounts that the model misclassified as posted by verified accounts.

The lower-left quadrant displays the number of false negatives: the number of videos posted by verified accounts that the model misclassified as posted by unverified accounts.

The lower-right quadrant displays the number of true positives: the number of videos posted by verified accounts that the model accurately classified as so.

A perfect model would yield all true negatives and true positives, and no false negatives or false positives.

Create a classification report that includes precision, recall, f1-score, and accuracy metrics to evaluate the performance of the logistic regression model.

In [58]:
```
# Create classification report for logistic regression model
target_labels = ["verified", "not verified"]
print(classification_report(y_test_final, y_pred, target_names=target_labels
```

```
              precision    recall  f1-score   support

    verified       0.74      0.46      0.57      4459
not verified       0.61      0.84      0.71      4483

    accuracy                           0.65      8942
   macro avg       0.67      0.65      0.64      8942
weighted avg       0.67      0.65      0.64      8942
```

Note: The classification report above shows that the logistic regression model achieved a precision of 61% and a recall of 84%, and it achieved an accuracy of 65%. Note that the precision and recall scores are taken from the "not verified" row of the output because that is the target class that we are most interested in predicting. The "verified" class has its own precision/recall metrics, and the weighted average represents the combined metrics for both classes of the target variable.

Task 4c. Interpret model coefficients

In [59]:
```
# Get the feature names from the model and the model coefficients (which rep
# Place into a DataFrame for readability
pd.DataFrame(data={"Feature Name":log_clf.feature_names_in_, "Model Coeffici
```

Out[59]:

	Feature Name	Model Coefficient
0	video_duration_sec	8.607893e-03
1	video_view_count	-2.132079e-06
2	video_share_count	5.930971e-06
3	video_download_count	-1.099775e-05
4	video_comment_count	-6.404235e-04
5	claim_status_opinion	3.908384e-04
6	author_ban_status_banned	-1.781741e-05
7	author_ban_status_under review	-9.682447e-07

Task 4d. Conclusion

1. What are the key takeaways from this project?
2. What results can be presented from this project?

Response:

Key takeaways:

- The dataset has a few strongly correlated variables, which might lead to multicollinearity issues when fitting a logistic regression model. We decided to drop `video_like_count` from the model building.
- Based on the logistic regression model, each additional second of the video is associated with 0.009 increase in the log-odds of the user having a verified status.
- The logistic regression model had not great, but acceptable predictive power: a precision of 61% is less than ideal, but a recall of 84% is very good. Overall accuracy is towards the lower end of what would typically be considered acceptable.

We developed a logistic regression model for verified status based on video features. The model had decent predictive power. Based on the estimated model coefficients from the logistic regression, longer videos tend to be associated with higher odds of the user being verified. Other video features have small estimated coefficients in the model, so their association with verified status seems to be small.