

# Activity\_Course 5 TikTok project lab

February 5, 2024

## 1 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.

## 2 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 demonstrate 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?

- The purpose of EDA before regression model building is to give the analyst and their team an idea of what data is relevant to their objective. In this case through our EDA we've determined that verified authors are more likely to post opinion pieces whereas non-verified

authors are likely to post claims. This is something we're going to investigate in this segment using a regression model.

**Part 2: Model Building and Evaluation** \* What resources do you find yourself using as you complete this stage?

- I will be using the pandas, seaborn, sklearn libraries primarily for the completion of this project.

**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.

## 3 Build a regression model

### 4 PACE stages

Throughout these project notebooks, you'll see references to the problem-solving framework PACE. The following notebook components are labeled with the respective PACE stage: Plan, Analyze, Construct, and Execute.

#### 4.1 PACE: Plan

Consider the questions in your PACE Strategy Document to reflect on the Plan stage.

##### 4.1.1 Task 1. Imports and loading

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

```
[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.

```

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

```

## 4.2 PACE: Analyze

Consider the questions in your PACE Strategy Document to reflect on the Analyze stage.

In this stage, consider the following question where applicable to complete your code response:

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

==> ENTER YOUR RESPONSE HERE

### 4.2.1 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.

```

[4]: # Display first few rows
data.head(10)

```

```

[4]:   # claim_status  video_id  video_duration_sec  \
0      1         claim  7017666017             59
1      2         claim  4014381136             32
2      3         claim  9859838091             31
3      4         claim  1866847991             25
4      5         claim  7105231098             19
5      6         claim  8972200955             35
6      7         claim  4958886992             16
7      8         claim  2270982263             41
8      9         claim  5235769692             50
9     10         claim  4660861094             45

      video_transcription_text  verified_status  \
0  someone shared with me that drone deliveries a...  not verified
1  someone shared with me that there are more mic...  not verified
2  someone shared with me that american industria...  not verified
3  someone shared with me that the metro of st. p...  not verified

```

4	someone shared with me that the number of busi...	not verified
5	someone shared with me that gross domestic pro...	not verified
6	someone shared with me that elvis presley has ...	not verified
7	someone shared with me that the best selling s...	not verified
8	someone shared with me that about half of the ...	not verified
9	someone shared with me that it would take a 50...	verified

	author_ban_status	video_view_count	video_like_count	video_share_count	\
0	under review	343296.0	19425.0	241.0	
1	active	140877.0	77355.0	19034.0	
2	active	902185.0	97690.0	2858.0	
3	active	437506.0	239954.0	34812.0	
4	active	56167.0	34987.0	4110.0	
5	under review	336647.0	175546.0	62303.0	
6	active	750345.0	486192.0	193911.0	
7	active	547532.0	1072.0	50.0	
8	active	24819.0	10160.0	1050.0	
9	active	931587.0	171051.0	67739.0	

	video_download_count	video_comment_count
0	1.0	0.0
1	1161.0	684.0
2	833.0	329.0
3	1234.0	584.0
4	547.0	152.0
5	4293.0	1857.0
6	8616.0	5446.0
7	22.0	11.0
8	53.0	27.0
9	4104.0	2540.0

Get the number of rows and columns in the dataset.

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

```
[5]: (19382, 12)
```

Get the data types of the columns.

```
[6]: # Get data types of columns
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

```

Get basic information about the dataset.

```
[7]: # Get basic information
data.describe()
```

```

[7]:
count    19382.000000    #    19382.000000    video_id    19382.000000    video_duration_sec    19084.000000    \
mean      9691.500000    5.627454e+09    32.421732    254708.558688
std       5595.245794    2.536440e+09    16.229967    322893.280814
min         1.000000    1.234959e+09     5.000000     20.000000
25%       4846.250000    3.430417e+09    18.000000    4942.500000
50%       9691.500000    5.618664e+09    32.000000    9954.500000
75%      14536.750000    7.843960e+09    47.000000   504327.000000
max      19382.000000    9.999873e+09    60.000000   999817.000000

count    19084.000000    video_like_count    19084.000000    video_share_count    19084.000000    \
mean      84304.636030    16735.248323    1049.429627
std     133420.546814    32036.174350    2004.299894
min         0.000000         0.000000         0.000000
25%       810.750000        115.000000         7.000000
50%       3403.500000        717.000000        46.000000
75%      125020.000000    18222.000000    1156.250000
max      657830.000000    256130.000000    14994.000000

count    19084.000000    video_comment_count
mean       349.312146
std       799.638865
min         0.000000
25%         1.000000
50%         9.000000
75%        292.000000

```

max 9599.000000

Generate basic descriptive statistics about the dataset.

```
[8]: # Generate basic descriptive stats
data.describe()
```

```
[8]:
```

	#	video_id	video_duration_sec	video_view_count	\
count	19382.000000	1.938200e+04	19382.000000	19084.000000	
mean	9691.500000	5.627454e+09	32.421732	254708.558688	
std	5595.245794	2.536440e+09	16.229967	322893.280814	
min	1.000000	1.234959e+09	5.000000	20.000000	
25%	4846.250000	3.430417e+09	18.000000	4942.500000	
50%	9691.500000	5.618664e+09	32.000000	9954.500000	
75%	14536.750000	7.843960e+09	47.000000	504327.000000	
max	19382.000000	9.999873e+09	60.000000	999817.000000	

	video_like_count	video_share_count	video_download_count	\
count	19084.000000	19084.000000	19084.000000	
mean	84304.636030	16735.248323	1049.429627	
std	133420.546814	32036.174350	2004.299894	
min	0.000000	0.000000	0.000000	
25%	810.750000	115.000000	7.000000	
50%	3403.500000	717.000000	46.000000	
75%	125020.000000	18222.000000	1156.250000	
max	657830.000000	256130.000000	14994.000000	

	video_comment_count
count	19084.000000
mean	349.312146
std	799.638865
min	0.000000
25%	1.000000
50%	9.000000
75%	292.000000
max	9599.000000

Check for and handle missing values.

```
[9]: # Check for missing values
data.isnull().sum(axis=0)
```

```
[9]: #
```

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
```

```
[10]: # Drop rows with missing values
data = data.dropna(axis=0).reset_index(drop = True)
```

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

```
[11]: # claim_status    video_id  video_duration_sec  \
0    1          claim  7017666017          59
1    2          claim  4014381136          32
2    3          claim  9859838091          31
3    4          claim  1866847991          25
4    5          claim  7105231098          19
5    6          claim  8972200955          35
6    7          claim  4958886992          16
7    8          claim  2270982263          41
8    9          claim  5235769692          50
9   10          claim  4660861094          45

          video_transcription_text  verified_status  \
0  someone shared with me that drone deliveries a...  not verified
1  someone shared with me that there are more mic...  not verified
2  someone shared with me that american industria...  not verified
3  someone shared with me that the metro of st. p...  not verified
4  someone shared with me that the number of busi...  not verified
5  someone shared with me that gross domestic pro...  not verified
6  someone shared with me that elvis presley has ...  not verified
7  someone shared with me that the best selling s...  not verified
8  someone shared with me that about half of the ...  not verified
9  someone shared with me that it would take a 50...  verified

author_ban_status  video_view_count  video_like_count  video_share_count  \
0      under review      343296.0      19425.0      241.0
1          active      140877.0      77355.0     19034.0
2          active      902185.0      97690.0      2858.0
3          active      437506.0     239954.0     34812.0
4          active       56167.0      34987.0      4110.0
5      under review      336647.0     175546.0     62303.0
6          active      750345.0     486192.0    193911.0
7          active      547532.0       1072.0        50.0
```

8	active	24819.0	10160.0	1050.0
9	active	931587.0	171051.0	67739.0

	video_download_count	video_comment_count
0	1.0	0.0
1	1161.0	684.0
2	833.0	329.0
3	1234.0	584.0
4	547.0	152.0
5	4293.0	1857.0
6	8616.0	5446.0
7	22.0	11.0
8	53.0	27.0
9	4104.0	2540.0

Check for and handle duplicates.

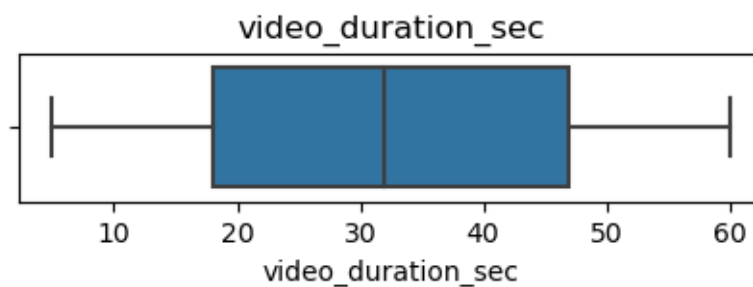
```
[55]: # Check for duplicates
data.duplicated().value_counts()
```

```
[55]: False    19084
      Name: count, dtype: int64
```

Check for and handle outliers.

```
[12]: # Create a boxplot to visualize distribution of `video_duration_sec`
plt.figure(figsize=(5,1))
plt.title('video_duration_sec')
sns.boxplot(x=data['video_duration_sec'])
```

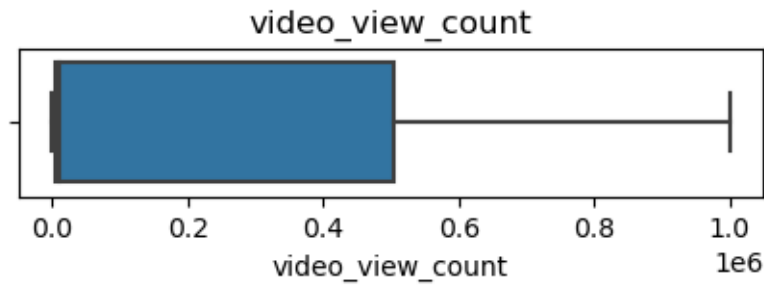
```
[12]: <Axes: title={'center': 'video_duration_sec'}, xlabel='video_duration_sec'>
```



```
[13]: # Create a boxplot to visualize distribution of `video_view_count`
plt.figure(figsize=(5,1))
plt.title('video_view_count')
sns.boxplot(x=data['video_view_count'])
```



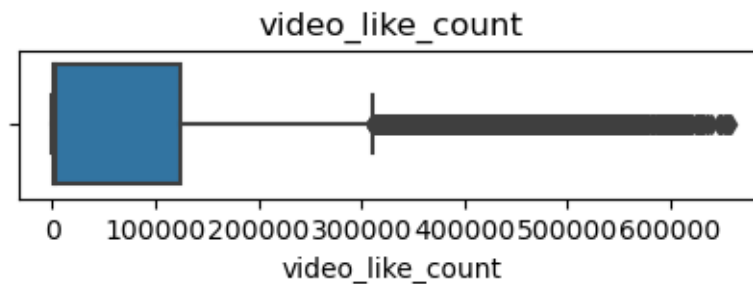
```
[13]: <Axes: title={'center': 'video_view_count'}, xlabel='video_view_count'>
```



```
[14]: # Create a boxplot to visualize distribution of `video_like_count`
```

```
plt.figure(figsize=(5,1))
plt.title("video_like_count")
sns.boxplot(x=data['video_like_count'])
```

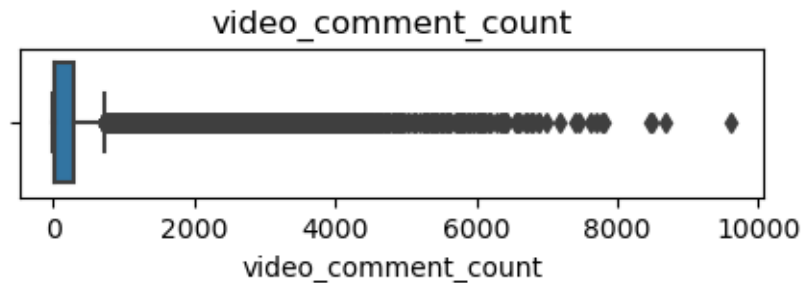
```
[14]: <Axes: title={'center': 'video_like_count'}, xlabel='video_like_count'>
```



```
[15]: # Create a boxplot to visualize distribution of `video_comment_count`
```

```
plt.figure(figsize=(5,1))
plt.title('video_comment_count')
sns.boxplot(x=data['video_comment_count'])
```

```
[15]: <Axes: title={'center': 'video_comment_count'}, xlabel='video_comment_count'>
```



```
[19]: # Check for and handle outliers for video_like_count

#25th percentile video_like_count
q1 = data['video_like_count'].quantile(0.25)
#75th percentile video_like_count
q3 = data['video_like_count'].quantile(0.75)
#calculate IQR video_like_count
iqr = q3 - q1
#Calculate median video_like_count
median = data['video_like_count'].median()
#Calculate thresholds video_like_count
median_outlier_threshold = median + 1.5*iqr
lower_limit = q1 - 1.5*iqr
upper_limit = q3 + 1.5*iqr

data.loc[data["video_like_count"] > upper_limit, "video_like_count"] = \
    ↪upper_limit

#25th percentile video_comment_count
percentile25 = data["video_comment_count"].quantile(0.25)
#75th percentile video_comment_count
percentile75 = data["video_comment_count"].quantile(0.75)
#calculate IQR video_comment_count
iqr = percentile75 - percentile25
#Calculate thresholds video_comment_count
upper_limit = percentile75 + 1.5 * iqr

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

Check class balance.

```
[20]: # Check class balance for video_comment_count

data['video_comment_count'].value_counts(normalize=True)
```

```
[20]: video_comment_count
      0.0      0.179941
      728.5    0.146143
      1.0      0.116433
      2.0      0.057483
      3.0      0.041291
      ...
      710.0    0.000052
      706.0    0.000052
      671.0    0.000052
      673.0    0.000052
      420.0    0.000052
      Name: proportion, Length: 720, 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.

```
[22]: # 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")
# This is taking an amount of majority not verified accounts and assigning the
# same amount of minority verified accounts. n_samples=len(data_majority)
data_minority_upsampled = resample(data_minority, replace=True,
# n_samples=len(data_majority), random_state=0)

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

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

```
[22]: 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.

```
[23]: # Get the average `video_transcription_text` length for claims and the average
# `video_transcription_text` length for opinions
```

```
data_upsampled[["verified_status", "video_transcription_text"]].
↳groupby(by="verified_status")[["video_transcription_text"]].agg(func=lambda
↳array: np.mean([len(text) for text in array]))
```

```
[23]:          video_transcription_text
verified_status
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.

```
[24]: # Extract the length of each `video_transcription_text` and add this as a
↳column to the dataframe
data_upsampled["text_length"] = data_upsampled["video_transcription_text"].
↳apply(func=lambda text: len(text))
```

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

```
[25]:  # claim_status    video_id  video_duration_sec  \
0  1          claim  7017666017          59
1  2          claim  4014381136          32
2  3          claim  9859838091          31
3  4          claim  1866847991          25
4  5          claim  7105231098          19

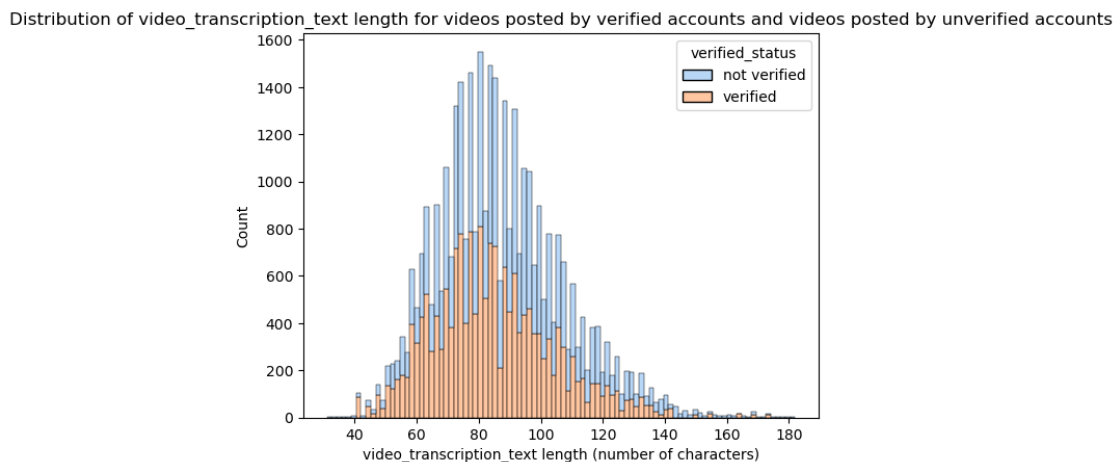
          video_transcription_text  verified_status  \
0  someone shared with me that drone deliveries a...  not verified
1  someone shared with me that there are more mic...  not verified
2  someone shared with me that american industria...  not verified
3  someone shared with me that the metro of st. p...  not verified
4  someone shared with me that the number of busi...  not verified

author_ban_status  video_view_count  video_like_count  video_share_count  \
0      under review          343296.0          19425.0           241.0
1           active          140877.0          77355.0          19034.0
2           active          902185.0          97690.0           2858.0
3           active          437506.0         239954.0          34812.0
4           active           56167.0          34987.0           4110.0

video_download_count  video_comment_count  text_length
0              1.0              0.0           97
1             1161.0             684.0          107
2              833.0             329.0          137
3             1234.0             584.0          131
4              547.0             152.0          128
```

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

```
[26]: # Visualize the distribution of `video_transcription_text` length for videos
      ↪ posted by verified accounts and videos posted by unverified accounts
      # Create two histograms in one plot
      sns.histplot(data=data_upsampled, stat="count", multiple="stack",
      ↪ x="text_length", kde=False, palette="pastel",
      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 posted by
      ↪ verified accounts and videos posted by unverified accounts")
      plt.show()
```



#### 4.2.2 Task 2b. Examine correlations

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

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

```
[29]:
```

	#	video_id	video_duration_sec	\
#	1.000000	-0.000853	-0.011729	
video_id	-0.000853	1.000000	0.011859	
video_duration_sec	-0.011729	0.011859	1.000000	
video_view_count	-0.697007	0.002554	0.013589	
video_like_count	-0.626385	0.005993	0.004494	
video_share_count	-0.504015	0.010515	0.002206	
video_download_count	-0.487096	0.008753	0.003989	

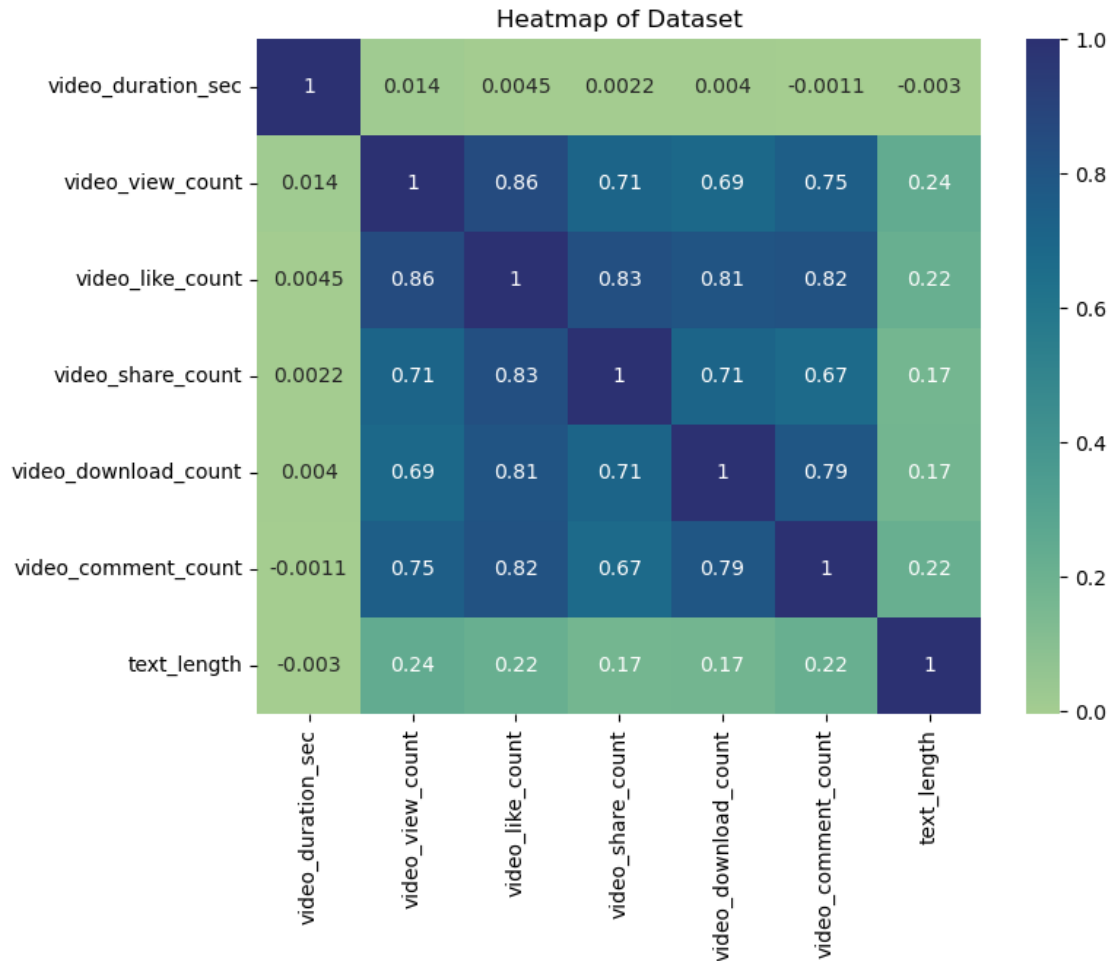
video_comment_count	-0.608773	0.012674	-0.001086
text_length	-0.193677	-0.007083	-0.002981

	video_view_count	video_like_count	video_share_count	\
#	-0.697007	-0.626385	-0.504015	
video_id	0.002554	0.005993	0.010515	
video_duration_sec	0.013589	0.004494	0.002206	
video_view_count	1.000000	0.856937	0.711313	
video_like_count	0.856937	1.000000	0.832146	
video_share_count	0.711313	0.832146	1.000000	
video_download_count	0.690048	0.805543	0.710117	
video_comment_count	0.748361	0.818032	0.671335	
text_length	0.244693	0.216693	0.171651	

	video_download_count	video_comment_count	text_length
#	-0.487096	-0.608773	-0.193677
video_id	0.008753	0.012674	-0.007083
video_duration_sec	0.003989	-0.001086	-0.002981
video_view_count	0.690048	0.748361	0.244693
video_like_count	0.805543	0.818032	0.216693
video_share_count	0.710117	0.671335	0.171651
video_download_count	1.000000	0.793668	0.173396
video_comment_count	0.793668	1.000000	0.217661
text_length	0.173396	0.217661	1.000000

Visualize a correlation heatmap of the data.

```
[30]: # 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_view_count",
                                "video_like_count", "video_share_count",
↪ "video_download_count", "video_comment_count", "text_length"]]
            .corr(numeric_only=True),
            annot=True,
            cmap='crest')
plt.title('Heatmap of 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.

**Question:** What variables are shown to be correlated in the heatmap? The above heatmap shows that the following pair of variables are strongly correlated: video\_view\_count and video\_like\_count (0.86 correlation coefficient).

### 4.3 PACE: Construct

After analysis and deriving variables with close relationships, it is time to begin constructing the model. Consider the questions in your PACE Strategy Document to reflect on the Construct stage.

#### 4.3.1 Task 3a. Select variables

Set your Y and X variables.

Select the outcome variable.

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

Select the features.

```
[85]: # Select features
# Select features
X = data_upsampled[["video_duration_sec", "claim_status", "author_ban_status",
↪ "video_view_count", "video_share_count", "video_download_count",
↪ "video_comment_count"]]

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

```
[85]:  video_duration_sec  claim_status  author_ban_status  video_view_count  \
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

    video_share_count  video_download_count  video_comment_count
0             241.0             1.0             0.0
1          19034.0          1161.0          684.0
2           2858.0           833.0          329.0
3          34812.0          1234.0          584.0
4           4110.0           547.0          152.0
```

#### 4.3.2 Task 3b. Train-test split

Split the data into training and testing sets.

```
[86]: # 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,
↪ random_state=0)
```

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

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

```
[89]: ((26826, 7), (8942, 7), (26826,), (8942,))
```

#### 4.3.3 Task 3c. Encode variables

Check the data types of the features.



```
[90]: # Check data types
X_train.dtypes
```

```
[90]: 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
```

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

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

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

```
[92]: 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.

```
[93]: # 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()
```

```
[93]:      claim_status  author_ban_status
33058      opinion          active
20491      opinion          active
25583      opinion          active
18474      opinion          active
27312      opinion          active
```

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

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

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

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

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

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

```
[99]: # Place encoded training features (which is currently an array) into a dataframe
X_train_encoded_df = pd.DataFrame(data=X_train_encoded, columns=X_encoder.
    ↳get_feature_names_out())

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

```
[99]:   claim_status_opinion  author_ban_status_banned  \
0                      1.0                      0.0
1                      1.0                      0.0
2                      1.0                      0.0
3                      1.0                      0.0
4                      1.0                      0.0

   author_ban_status_under review
0                      0.0
1                      0.0
2                      0.0
3                      0.0
4                      0.0
```

```
[100]: # Display first few rows of `X_train` with `claim_status` and
    ↳ `author_ban_status` columns dropped (since these features are being
    ↳ transformed to numeric)
X_train.drop(columns=["claim_status", "author_ban_status"]).head()
```

```
[100]:   video_duration_sec  video_view_count  video_share_count  \
33058                33            2252.0                23.0
20491                52            6664.0               550.0
25583                37            6327.0               257.0
```

18474	57	1702.0	28.0
27312	21	3842.0	101.0

	video_download_count	video_comment_count
33058	4.0	0.0
20491	53.0	2.0
25583	3.0	0.0
18474	0.0	0.0
27312	1.0	0.0

```
[101]: # Concatenate `X_train` and `X_train_encoded_df` to form the final dataframe
        ↪ for training data (`X_train_final`)
# Note: Using `.reset_index(drop=True)` to reset the index in X_train after
        ↪ dropping `claim_status` and `author_ban_status`,
# 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_ban_status"]), X_train_encoded_df], axis=1)

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

```
[101]: video_duration_sec  video_view_count  video_share_count  \
0                33            2252.0            23.0
1                52            6664.0           550.0
2                37            6327.0           257.0
3                57            1702.0            28.0
4                21            3842.0           101.0

        video_download_count  video_comment_count  claim_status_opinion  \
0                4.0            0.0            1.0
1               53.0            2.0            1.0
2                3.0            0.0            1.0
3                0.0            0.0            1.0
4                1.0            0.0            1.0

        author_ban_status_banned  author_ban_status_under review
0                0.0            0.0
1                0.0            0.0
2                0.0            0.0
3                0.0            0.0
4                0.0            0.0
```

Check the data type of the outcome variable.

```
[102]: # Check data type of outcome variable
y_train.dtype
```

```
[102]: dtype('O')
```

```
[103]: # Get unique values of outcome variable
y_train.unique()
```

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

As 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.

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

```
[105]: # Encode the training outcome variable
# Notes:
#   - Adjusting the shape of `y_train` before passing into `.fit_transform()`,
#     ↳ since it takes in 2D array
#   - Using `.ravel()` to flatten the array returned by `.fit_transform()`, so
#     ↳ that it can be used later to train the model
y_train_final = y_encoder.fit_transform(y_train.values.reshape(-1, 1)).ravel()

# Display the encoded training outcome variable
y_train_final
```

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

#### 4.3.4 Task 3d. Model building

Construct a model and fit it to the training set.

```
[106]: # 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,
↳ y_train_final)
```

### 4.4 PACE: Execute

Consider the questions in your PACE Strategy Document to reflect on the Execute stage.

#### 4.4.1 Task 4a. Results and evaluation

Evaluate your model.

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

```
[107]: # 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()
```

```
[107]:      claim_status  author_ban_status
21061      opinion          active
31748      opinion          active
20197        claim          active
5727        claim          active
11607      opinion          active
```

```
[76]: # 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
```

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

```
[108]: # Place encoded testing features (which is currently an array) into a dataframe
X_test_encoded_df = pd.DataFrame(data=X_test_encoded, columns=X_encoder.
    ↪get_feature_names_out())

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

```
[108]:      claim_status_opinion  author_ban_status_banned  \
0                1.0                0.0
1                1.0                0.0
2                0.0                0.0
3                0.0                0.0
4                1.0                0.0

      author_ban_status_under review
0                0.0
1                0.0
2                0.0
3                0.0
4                0.0
```

```
[78]:
```

```
# Display first few rows of `X_test` with `claim_status` and
↳ `author_ban_status` columns dropped (since these features are being
↳ transformed to numeric)
X_test.drop(columns=["claim_status", "author_ban_status"]).head()
```

```
[78]:      video_duration_sec  video_view_count  video_share_count  \
21061                41           2118.0           57.0
31748                27           5701.0           157.0
20197                31          449767.0          75385.0
5727                 19          792813.0          56597.0
11607                54           2044.0            68.0

      video_download_count  video_comment_count
21061                   5.0                   2.0
31748                   1.0                   0.0
20197                 5956.0                  728.5
5727                  5146.0                  728.5
11607                  19.0                   2.0
```

```
[109]: # Concatenate `X_test` and `X_test_encoded_df` to form the final dataframe for
↳ training data (`X_test_final`)
# Note: Using `.reset_index(drop=True)` to reset the index in `X_test` after
↳ dropping `claim_status`, and `author_ban_status`,
# so that the indices align with those in `X_test_encoded_df` and
↳ `test_count_df`
X_test_final = pd.concat([X_test.drop(columns=["claim_status",
↳ "author_ban_status"]).reset_index(drop=True), X_test_encoded_df], axis=1)

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

```
[109]:      video_duration_sec  video_view_count  video_share_count  \
0                41           2118.0           57.0
1                27           5701.0           157.0
2                31          449767.0          75385.0
3                19          792813.0          56597.0
4                54           2044.0            68.0

      video_download_count  video_comment_count  claim_status_opinion  \
0                   5.0                   2.0                1.0
1                   1.0                   0.0                1.0
2                 5956.0                  728.5                0.0
3                 5146.0                  728.5                0.0
4                  19.0                   2.0                1.0

      author_ban_status_banned  author_ban_status_under review
0                   0.0                0.0
```

1	0.0	0.0
2	0.0	0.0
3	0.0	0.0
4	0.0	0.0

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

```
[110]: # Use the logistic regression model to get predictions on the encoded testing
      ↪ set
      y_pred = log_clf.predict(X_test_final)
```

Display the predictions on the encoded testing set.

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

```
[111]: array([1., 1., 0., ..., 1., 0., 1.]
```

Display the true labels of the testing set.

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

```
[112]: 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.

```
[113]: # Encode the testing outcome variable
      # Notes:
      #   - Adjusting the shape of `y_test` before passing into `.transform()`, since
      ↪ it takes in 2D array
      #   - Using `.ravel()` to flatten the array returned by `.transform()`, so that
      ↪ it can be used later to compare with predictions
      y_test_final = y_encoder.transform(y_test.values.reshape(-1, 1)).ravel()

      # Display the encoded testing outcome variable
      y_test_final
```

```
[113]: 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.

```
[114]: # Get shape of each training and testing set
X_train_final.shape, y_train_final.shape, X_test_final.shape, y_test_final.shape
```

```
[114]: ((26826, 8), (26826,), (8942, 8), (8942,))
```

#### 4.4.2 Task 4b. Visualize model results

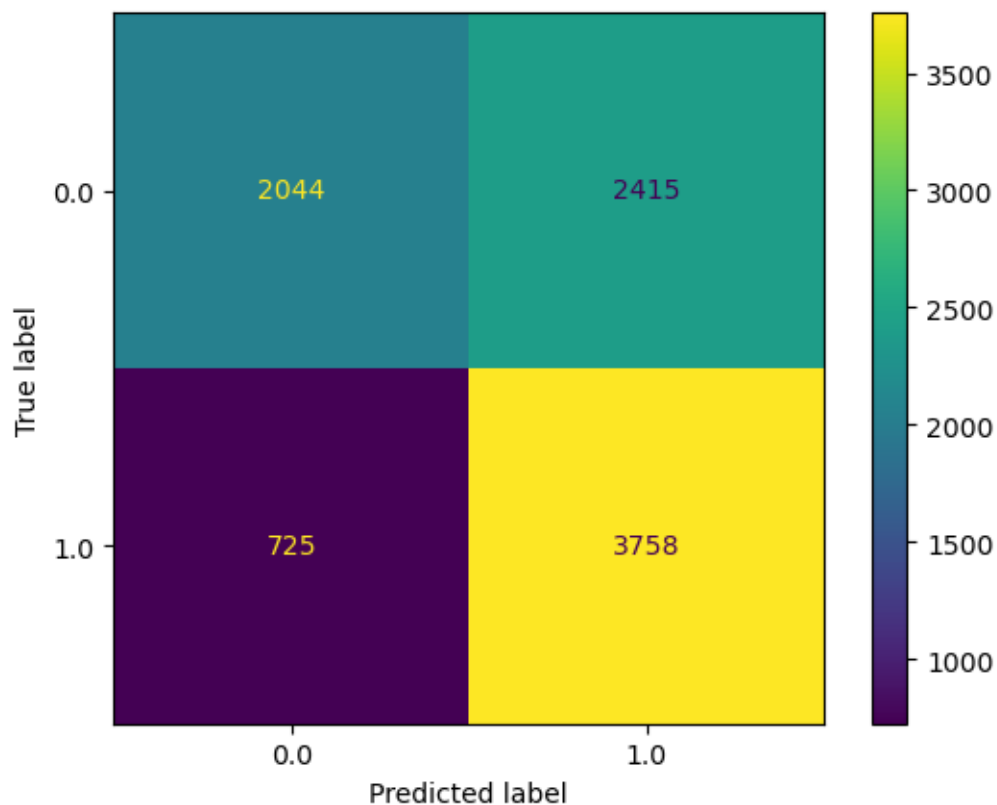
Create a confusion matrix to visualize the results of the logistic regression model.

```
[117]: # 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=log_clf.classes_)
# Plot confusion matrix
log_disp.plot()

# Display plot
plt.show()
```





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

```
[118]: # Create a classification report
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

#### 4.4.3 Task 4c. Interpret model coefficients

```
[119]: # Get the feature names from the model and the model coefficients (which
      ↪ represent log-odds ratios)
      # Place into a DataFrame for readability
      pd.DataFrame(data={"Feature Name":log_clf.feature_names_in_, "Model_
      ↪Coefficient":log_clf.coef_[0]})
```

```
[119]:
```

	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

#### 4.4.4 Task 4d. Conclusion

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

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.

**Congratulations!** You've completed this lab. However, you may not notice a green check mark next to this item on Coursera's platform. Please continue your progress regardless of the check mark. Just click on the "save" icon at the top of this notebook to ensure your work has been logged.