Activity_Course 5 TikTok project lab

February 1, 2025

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 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.

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

4.1.1 Task 1. Imports and loading

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

[1]:

```python
# Import packages for data manipulation
import pandas as pd
import numpy as np

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

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

# 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, 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.

[2]:

```
# Load dataset into dataframe
data = pd.read_csv("tiktok_dataset.csv")
```

4.2 PACE: Analyze

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?

Basic problems with data that need fixed before a model can read and process the dataset are missing values and different variables having different information types (categorical vs numerical). Additionally, an assumption of multiple linear regression models is no multicollinearity, so that assumption will need checked and adjusted for should the data prove problematic.

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.

[3]:

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

[3]:

```
   # 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
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
```

Get the number of rows and columns in the dataset.

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

[4]:
```
(19382, 12)
```

Get the data types of the columns.

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

[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.

[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.

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

[7]:
```
                  #      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.

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

[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
```

[9]:

```
# Drop rows with missing values
data = data.dropna(axis = 0)
```

[10]:

```
# Display first few rows after handling missing values
data.head()
```

[10]:

```
   # 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
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
```

Check for and handle duplicates.

[11]:

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

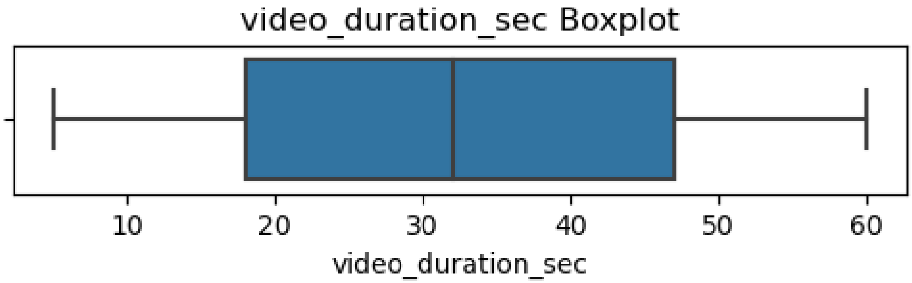
[11]:

```
0
```

Check for and handle outliers.

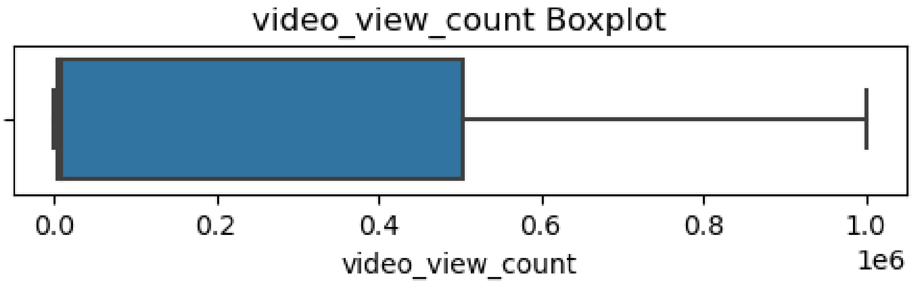
[12]:

```
# Create a boxplot to visualize distribution of `video_duration_sec`
plt.figure(figsize = (6,1))
plt.title('video_duration_sec Boxplot')
sns.boxplot(x = data['video_duration_sec'])

plt.show()
```



[13]:

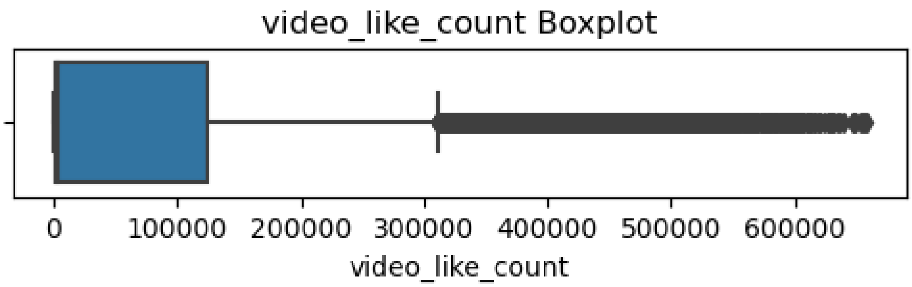
```
# Create a boxplot to visualize distribution of `video_view_count`
plt.figure(figsize = (6,1))
plt.title('video_view_count Boxplot')
sns.boxplot(x = data['video_view_count'])

plt.show()
```



[14]:

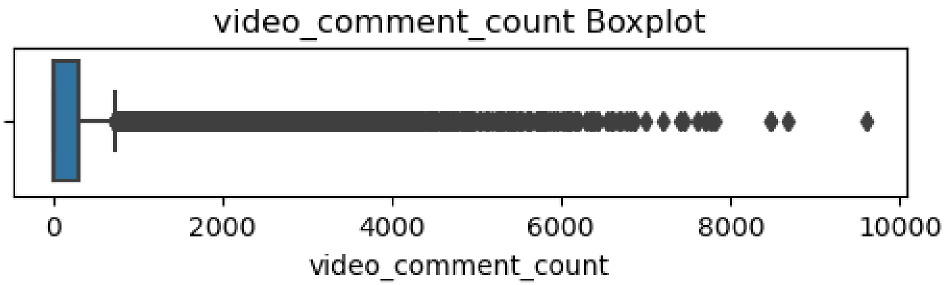
```
# Create a boxplot to visualize distribution of `video_like_count`
plt.figure(figsize = (6,1))
plt.title('video_like_count Boxplot')
```

```
sns.boxplot(x = data['video_like_count'])

plt.show()
```



[15]:

```
# Create a boxplot to visualize distribution of `video_comment_count`
plt.figure(figsize = (6,1))
plt.title('video_comment_count Boxplot')
sns.boxplot(x = data['video_comment_count'])

plt.show()
```



[16]:

```
# Check for and handle outliers for video_like_count
quart = data['video_like_count'].quantile(.25)
threeQuart = data['video_like_count'].quantile(.75)

IQR = threeQuart - quart
upperL = threeQuart + 1.5 * IQR
data.loc[data['video_like_count'] > upperL, 'video_like_count'] = upperL
```

[17]:

```
#Check for and handle outliers for video_comment_count
quart = data['video_comment_count'].quantile(.25)
threeQuart = data['video_comment_count'].quantile(.75)

IQR = threeQuart - quart
```

```
upperL = threeQuart + 1.5 * IQR
data.loc[data['video_comment_count'] > upperL, 'video_comment_count'] = upperL
```

Check class balance of the target variable. Remember, the goal is to predict whether the user of a given post is verified or unverified.

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

```
[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.

```
[19]: # Use resampling to create class balance in the outcome variable, if needed
      # Identify data points from majority and minority classes
      majority = data[data['verified_status'] == 'not verified']
      minority = data[data['verified_status'] == 'verified']

      # Upsample the minority class (which is "verified")
      upsample = resample(minority,
                          replace = True,
                          n_samples = len(majority),
                          random_state = 0)

      # Combine majority class with upsampled minority class
      newData = pd.concat([majority, upsample]).reset_index(drop=True)

      # Display new class counts
      newData['verified_status'].value_counts()
```

```
[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.

```
[20]: # Get the average `video_transcription_text` length for claims and the average␣
       ↪`video_transcription_text` length for opinions
      newData[['verified_status', 'video_transcription_text']].groupby(
          'verified_status')[['video_transcription_text']].agg(
    func = lambda array: np.mean([len(text) for text in array]))
```

[20]:

```
                  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.

[21]:

```
# Extract the length of each `video_transcription_text` and add this as a␣
 ↪column to the dataframe
newData['text_length'] = newData['video_transcription_text'].apply(func =␣
 ↪lambda text: len(text))
```

[22]:

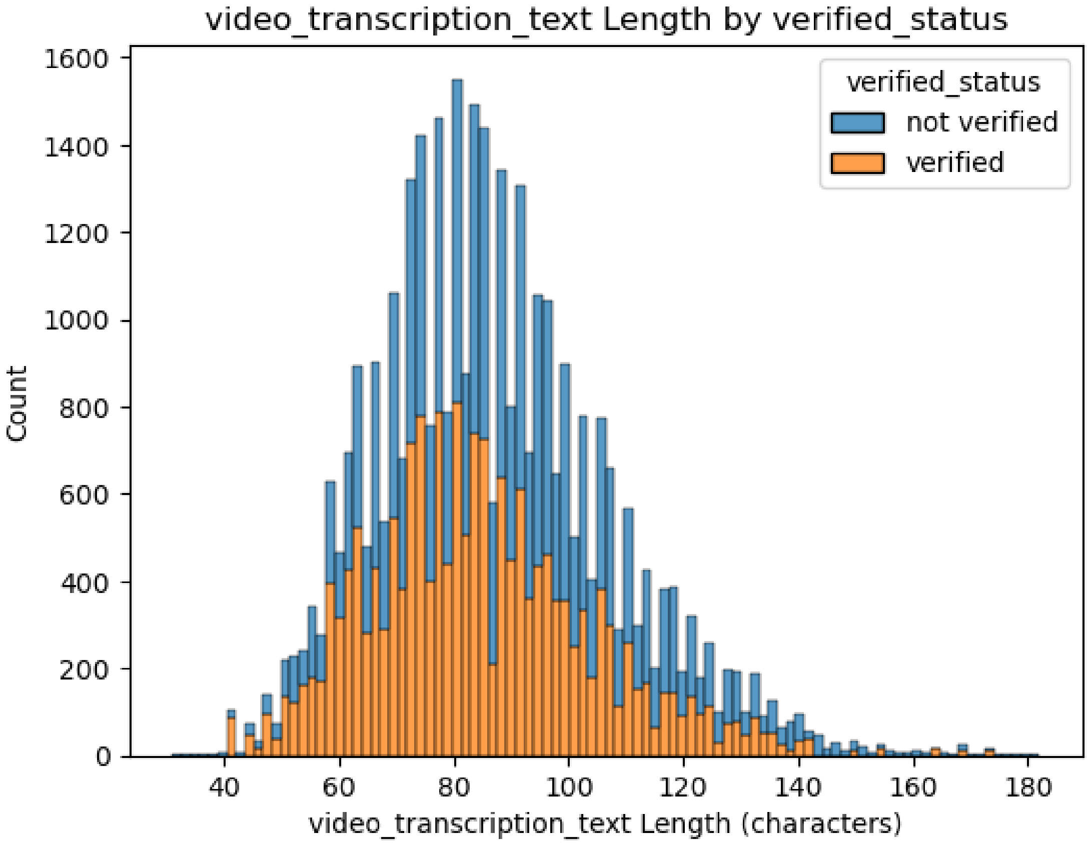
```
# Display first few rows of dataframe after adding new column
newData.head()
```

[22]:

```
   # 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.

[23]:
```
# 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(newData,
             stat = 'count',
             multiple = 'stack',
             x = 'text_length',
             kde = False,
             hue = 'verified_status',
             legend = True)
plt.title('video_transcription_text Length by verified_status')
plt.xlabel('video_transcription_text Length (characters)')
plt.ylabel('Count')
plt.show()
```



4.2.2 Task 2b. Examine correlations

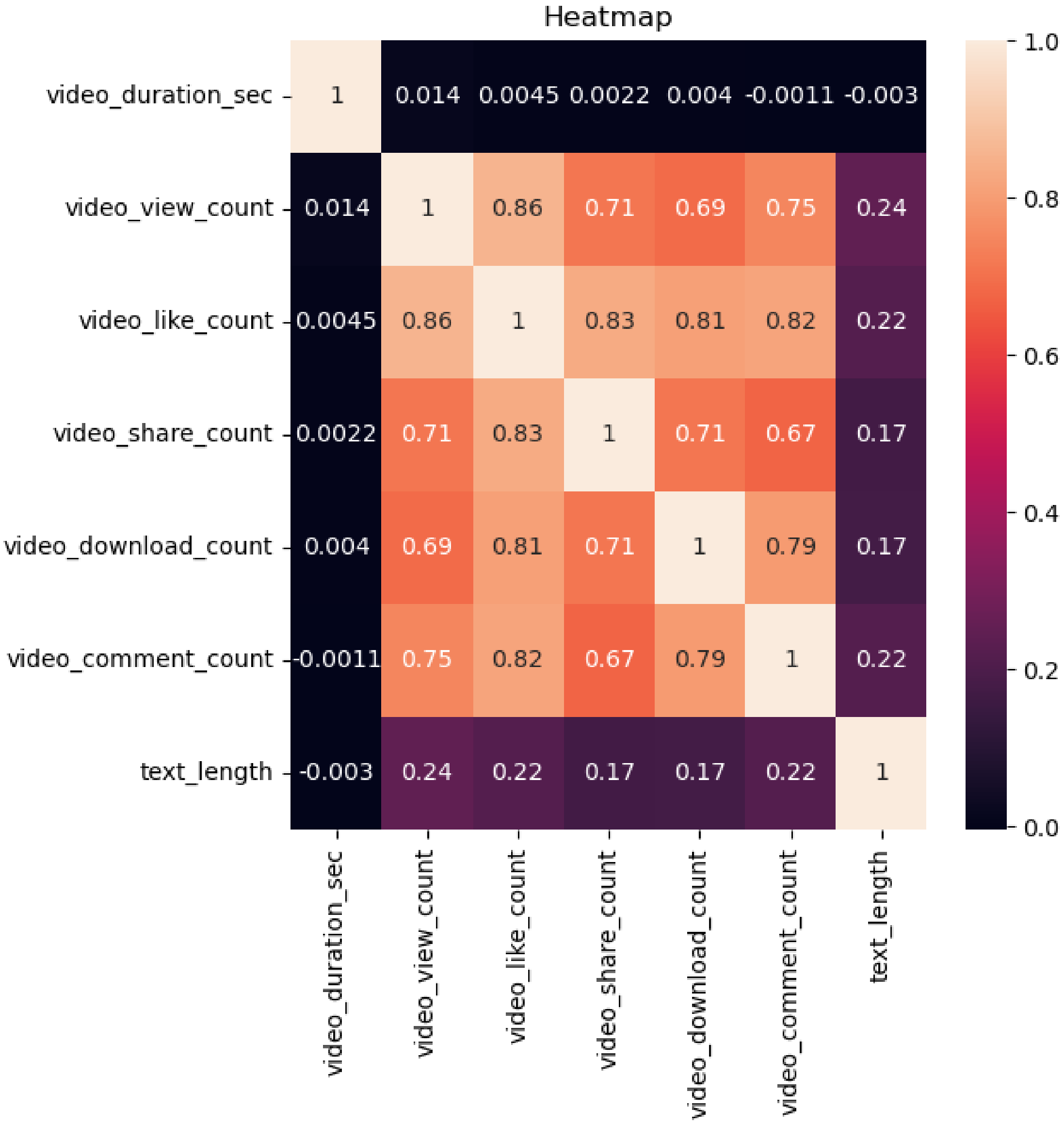
Next, code a correlation matrix to help determine most correlated variables.

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

[24]:
```
                            #  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.

[25]:
```
# Create a heatmap to visualize how correlated variables are
plt.figure(figsize=(6,6))
sns.heatmap(
    newData[['video_duration_sec', 'author_ban_status', 'video_view_count',␣
 ↪'claim_status','video_like_count',
            'video_share_count', 'video_download_count', 'video_comment_count',␣
 ↪'text_length']].corr(numeric_only = True),
    annot = True)
plt.title('Heatmap')
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? While many of the variables show significant correlation, discarding many of them could result in a significant loss of information that would be useful to the model. So we will extract the variable that correlates most with the others, video_like_count, to try to further satisfy the multicollinearity assumption of multiple linear regression.

4.3 PACE: Construct

4.3.1 Task 3a. Select variables

Set your Y and X variables.

Select the outcome variable.

```
[26]: # Select outcome variable
      y = newData['verified_status']
```

Select the features.

```
[27]: # Select features
      X = newData[['author_ban_status', 'claim_status', 'video_comment_count',␣
        ↪'video_download_count', 'video_duration_sec',
                    'video_share_count', 'video_view_count']]

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

```
[27]:   author_ban_status claim_status  video_comment_count  video_download_count  \
      0      under review        claim                  0.0                   1.0
      1            active        claim                684.0                1161.0
      2            active        claim                329.0                 833.0
      3            active        claim                584.0                1234.0
      4            active        claim                152.0                 547.0

         video_duration_sec  video_share_count  video_view_count
      0                  59              241.0          343296.0
      1                  32            19034.0          140877.0
      2                  31             2858.0          902185.0
      3                  25            34812.0          437506.0
      4                  19             4110.0           56167.0
```

4.3.2 Task 3b. Train-test split

Split the data into training and testing sets.

```
[28]: # Split the data into training and testing sets
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .25,␣
        ↪random_state = 0)
```

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

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

```
[29]: ((26826, 7), (8942, 7), (26826,), (8942,))
```

4.3.3 Task 3c. Encode variables

Check the data types of the features.

[30]:
```
# Check data types
X_train.dtypes
```

[30]:
```
author_ban_status        object
claim_status             object
video_comment_count     float64
video_download_count    float64
video_duration_sec        int64
video_share_count       float64
video_view_count        float64
dtype: object
```

[31]:
```
# Get unique values in `claim_status`
X_train['claim_status'].unique()
```

[31]:
```
array(['opinion', 'claim'], dtype=object)
```

[32]:
```
# Get unique values in `author_ban_status`
X_train['author_ban_status'].unique()
```

[32]:
```
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.

[33]:
```
# Select the training features that needs to be encoded
X_train_encode = X_train[['claim_status', 'author_ban_status']]

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

[33]:
```
      claim_status author_ban_status
33058      opinion            active
20491      opinion            active
25583      opinion            active
18474      opinion            active
27312      opinion            active
```

[34]:
```
# Set up an encoder for one-hot encoding the categorical features
X_encoder = OneHotEncoder(drop = 'first', sparse_output = False)
```

[35]:

```
# Fit and transform the training features using the encoder
X_train_encoded = X_encoder.fit_transform(X_train_encode)
```

[36]:

```
# Get feature names from encoder
X_encoder.get_feature_names_out()
```

[36]:

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

[37]:

```
# Display first few rows of encoded training features
X_train_encoded
```

[37]:

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

[38]:

```
# Place encoded training features (which is currently an array) into a dataframe
X_train_encoded_df = pd.DataFrame(X_train_encoded, columns = X_encoder.
 ↪get_feature_names_out())

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

[38]:

```
   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
```

[39]:

```
# 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()
```

```
[39]:        video_comment_count  video_download_count  video_duration_sec  \
      33058                  0.0                   4.0                  33
      20491                  2.0                  53.0                  52
      25583                  0.0                   3.0                  37
      18474                  0.0                   0.0                  57
      27312                  0.0                   1.0                  21

             video_share_count  video_view_count
      33058               23.0            2252.0
      20491              550.0            6664.0
      25583              257.0            6327.0
      18474               28.0            1702.0
      27312              101.0            3842.0
```

```
[40]: # 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']).reset_index(drop = True), X_train_encoded_df], axis =␣
       ↪1)

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

```
[40]:    video_comment_count  video_download_count  video_duration_sec  \
      0                  0.0                   4.0                  33
      1                  2.0                  53.0                  52
      2                  0.0                   3.0                  37
      3                  0.0                   0.0                  57
      4                  0.0                   1.0                  21

         video_share_count  video_view_count  claim_status_opinion  \
      0               23.0            2252.0                   1.0
      1              550.0            6664.0                   1.0
      2              257.0            6327.0                   1.0
      3               28.0            1702.0                   1.0
      4              101.0            3842.0                   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.

[41]:
```
# Check data type of outcome variable
y_train.dtype
```

[41]:
```
dtype('O')
```

[42]:
```
# Get unique values of outcome variable
y_train.unique()
```

[42]:
```
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.

[43]:
```
# Set up an encoder for one-hot encoding the categorical outcome variable
y_encoder = OneHotEncoder(drop = 'first', sparse_output = False)
```

[44]:
```
# 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
```

[44]:
```
array([1., 1., 1., …, 1., 1., 0.])
```

4.3.4 Task 3d. Model building

Construct a model and fit it to the training set.

[45]:
```
# 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

4.4.1 Taks 4a. Results and evaluation

Evaluate your model.

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

[46]:

```
# Select the testing features that needs to be encoded
X_test_encode = X_test[['claim_status', 'author_ban_status']]

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

[46]:

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

[47]:

```
# Transform the testing features using the encoder
X_test_encoded = X_encoder.transform(X_test_encode)

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

[47]:

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

[48]:

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

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

[48]:

```
   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
```

[49]:
```
# 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()
```

[49]:
```
       video_comment_count  video_download_count  video_duration_sec  \
21061                  2.0                   5.0                  41
31748                  0.0                   1.0                  27
20197                728.5                5956.0                  31
5727                 728.5                5146.0                  19
11607                  2.0                  19.0                  54

       video_share_count  video_view_count
21061               57.0            2118.0
31748              157.0            5701.0
20197            75385.0          449767.0
5727             56597.0          792813.0
11607               68.0            2044.0
```

[50]:
```
# 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()
```

[50]:
```
   video_comment_count  video_download_count  video_duration_sec  \
0                  2.0                   5.0                  41
1                  0.0                   1.0                  27
2                728.5                5956.0                  31
3                728.5                5146.0                  19
4                  2.0                  19.0                  54

   video_share_count  video_view_count  claim_status_opinion  \
0               57.0            2118.0                   1.0
1              157.0            5701.0                   1.0
2            75385.0          449767.0                   0.0
3            56597.0          792813.0                   0.0
4               68.0            2044.0                   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.

```
[51]: # 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.

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

```
[52]: array([1., 1., 0., …, 1., 0., 1.])
```

Display the true labels of the testing set.

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

```
[53]: 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.

```
[54]: # 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
```

```
[54]: 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.

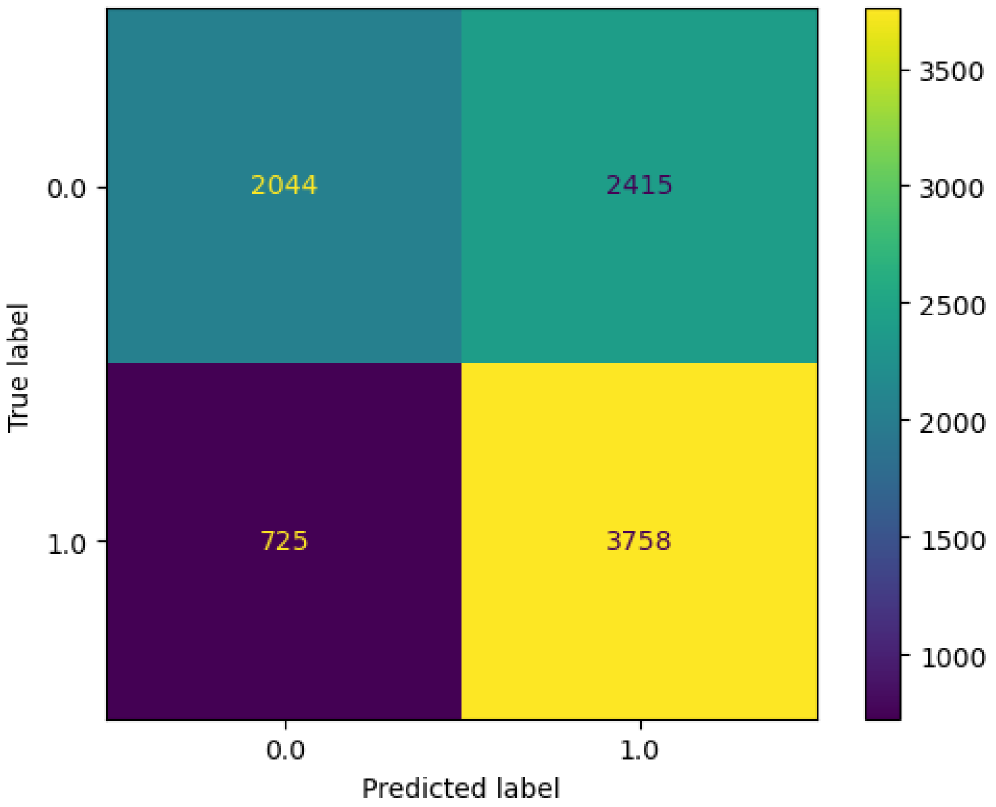
```
[55]: # Get shape of each training and testing set
      X_train_final.shape, y_train_final.shape, X_test_final.shape, y_test_final.shape
```

```
[55]: ((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.

```
[56]: # 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.

[57]:

```
# Create a classification report
labels = ['not verified', 'verified']
print(classification_report(y_test_final, y_pred, target_names = labels))
```

```
              precision    recall  f1-score   support

not verified       0.74      0.46      0.57      4459
    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

[58]:

```
# 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]})
```

[58]:

```
                   Feature Name  Model Coefficient
0           video_comment_count      -6.404235e-04
1          video_download_count      -1.099775e-05
2            video_duration_sec       8.607893e-03
3             video_share_count       5.930971e-06
4              video_view_count      -2.132079e-06
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?

Our logistic regression model found that each additional second of a tiktok resulted in a .009 increase in the log odds of a tiktok coming from a verified account. The mean tiktok has 32 seconds, and 32 * .009 = .288, so the tiktoks are long enough for tiktok duration to have a significant impact on the log odds of an account being verified. The model's performance wasn't phenomenal, but with an 'verified' predictive precision of .61, recall of .84, and an accuracy of .65, the model demonstrated acceptable predictive power.

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.