TikTok project

September 7, 2024

1 TikTok Project

Regression Analysis: Simplify complex data relationships

In this project we will assume that i'm 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.

TikTok's Operations Lead, 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, i 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.

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 project has three parts:*

Part 1: EDA & Checking Model Assumptions

Part 2: Model Building and Evaluation

Part 3: Interpreting Model Results

2 Building a regression model

2.1 PACE: Plan

2.1.1 1. Imports and loading

[3]:
```
# 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.

[4]:
```
# Load dataset into dataframe
data = pd.read_csv("tiktok_dataset.csv")
```

2.2 PACE: Analyze

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.

2.2.1 2a. Exploring data with EDA

We will analyze the data and checking for and handling missing values and duplicates.

The first five rows of the dataframe.

[5]:
```
data.head(5)
```

[5]:

	#	claim_status	video_id	video_duration_sec	video_transcription_text	verified_status	author_ban_status	video_view_count	video_like_count	video_share_count
0	1	claim	7017666017	59	someone shared with me that drone deliveries a…	not verified	under review	343296.0	19425.0	241.0
1	2	claim	4014381136	32	someone shared with me that there are more mic…	not verified	active	140877.0	77355.0	19034.0
2	3	claim	9859838091	31	someone shared with me that american industria…	not verified	active	902185.0	97690.0	2858.0
3	4	claim	1866847991	25	someone shared with me that the metro of st. p…	not verified				
4	5	claim	7105231098	19	someone shared with me that the number of busi…	not verified				

```
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
```

The number of rows and columns in the dataset.

[6]:
```
data.shape
```

[6]:
```
(19382, 12)
```

The data types of the columns.

[7]:
```
data.dtypes
```

[7]:
```
#                            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
```

Basic information about the dataset.

[8]:
```
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
```

Basic descriptive statistics about the dataset.

[9]:
```
data.describe()
```

[9]:

	#	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

Checking for and handling missing values.

[10]:
```
# Check for missing values
data.isna().sum()
```

[10]:
```
#                           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
```

There are a lot of missing values.

[11]:
```
# Drop rows with missing values
data = data.dropna(axis=0)
```

[12]:
```
# Displaying first few rows after handling missing values
data.head()
```

[12]:

	#	claim_status	video_id	video_duration_sec	video_transcription_text	verified_status	author_ban_status	video_view_count	video_like_count	video_share_count	video_download_count	video_comment_count
0	1	claim	7017666017	59	someone shared with me that drone deliveries a…	not verified	under review	343296.0	19425.0	241.0	1.0	0.0
1	2	claim	4014381136	32	someone shared with me that there are more mic…	not verified	active	140877.0	77355.0	19034.0	1161.0	684.0
2	3	claim	9859838091	31	someone shared with me that american industria…	not verified	active	902185.0	97690.0	2858.0	833.0	329.0
3	4	claim	1866847991	25	someone shared with me that the metro of st. p…	not verified	active	437506.0	239954.0	34812.0		
4	5	claim	7105231098	19	someone shared with me that the number of busi…	not verified	active	56167.0	34987.0	4110.0		

```
3                1234.0                584.0
4                 547.0                152.0
```

Checking for and handling duplicates.

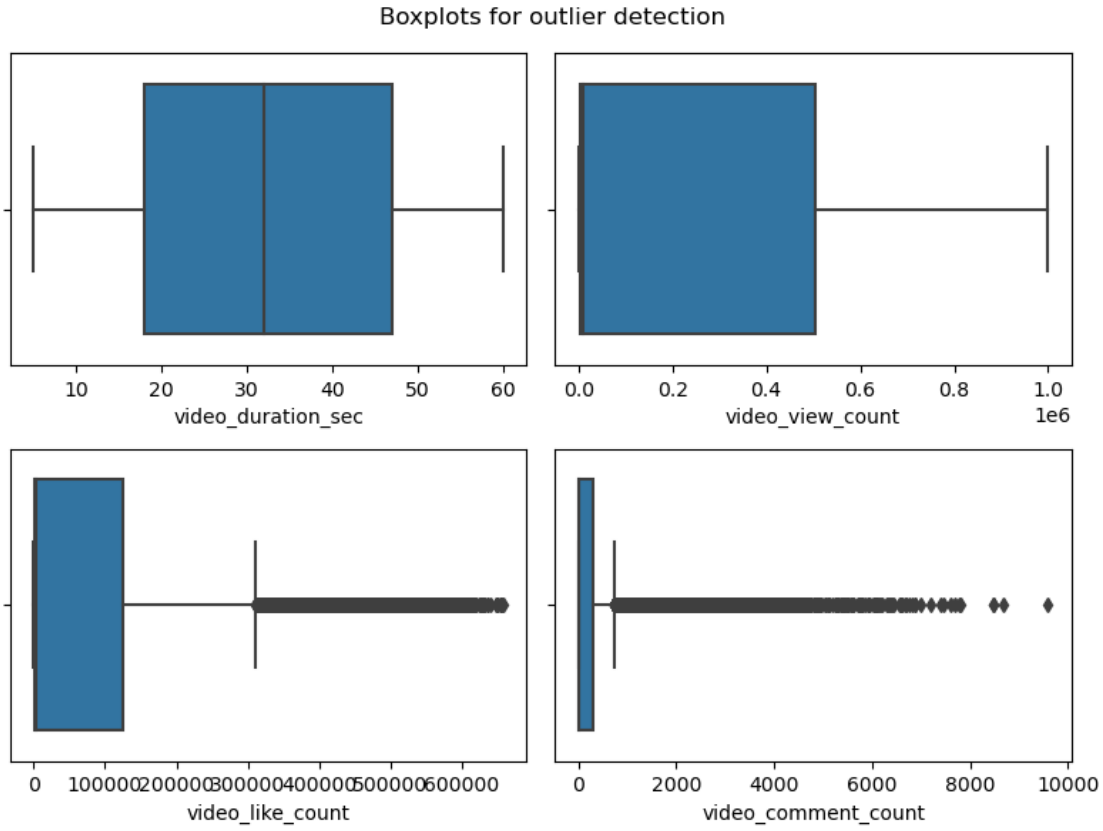
[13]:
```
data.duplicated().sum()
```

[13]:
```
0
```

There are no duplicates.

Checking for and handling outliers.

[14]:
```
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(8, 6))
fig.suptitle('Boxplots for outlier detection')
sns.boxplot(ax=axes[0][0], x=data['video_duration_sec'])
sns.boxplot(ax=axes[0][1], x=data['video_view_count'])
sns.boxplot(ax=axes[1][0], x=data['video_like_count'])
sns.boxplot(ax=axes[1][1], x=data['video_comment_count'])
plt.tight_layout()
plt.show();
```



`video_like_count` and `video_comment_count` contain outliers.

Imputations

Now We will impute the maximum value as Q3 + (1.5 * IQR).

[19]:
```
# 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_limit
```

[20]:
```
# 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"] =␣
 ↪upper_limit
```

Checking class balance.

[21]:
```
# Checking class balance for video_comment_count
data["verified_status"].value_counts(normalize=True)
```

[21]:
```
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.

Feature engineering We will use resampling to create class balance in the outcome variable, if needed.

[29]:
```
# Data points from majority and minority classes
data_majority = data[data['verified_status'] == 'not verified']
data_minority = data[data['verified_status'] == 'verified']

# Upsampling the minority class (which is "verified")
```

```
data_minority_upsampled = resample(data_minority, replace=True,␣
 ↪n_samples=len(data_majority), random_state=0)
# Combining 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()
```

[29]:
```
verified_status
not verified    17884
verified        17884
Name: count, dtype: int64
```

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.

[34]:
```
#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]))
data_upsampled.groupby('verified_status')[['video_transcription_text']].
 ↪agg(func=lambda array: np.mean([len(text) for text in array]))
```

[34]:

verified_status	video_transcription_text
not verified	89.401141
verified	84.569559

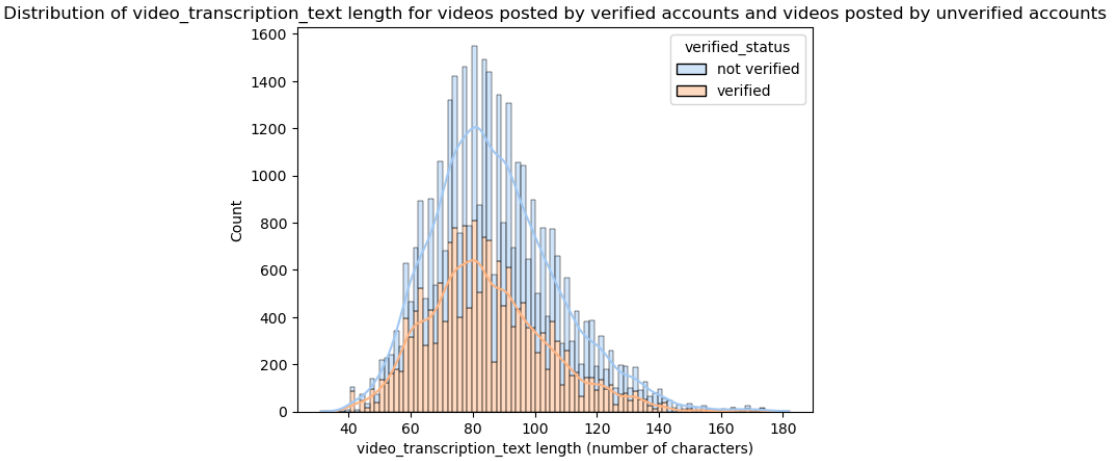
Extracting the length of each `video_transcription_text` and adding this as a column to the dataframe, so that it can be used as a potential feature in the model.

[44]:
```
data_upsampled['text_length'] = data_upsampled['video_transcription_text'].str.
 ↪len()
data_upsampled['text_length'].head()
```

[44]:
```
0     97
1    107
2    137
3    131
4    128
Name: text_length, dtype: int64
```

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

[48]:
```
sns.histplot(data=data_upsampled, stat="count", multiple="stack",␣
 ↪x="text_length", kde=True, 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()
```



2.2.2 2b. Examining correlations

Correlation matrix to help determine most correlated variables.

[49]:
```
data_upsampled.corr(numeric_only=True)
```

[49]:

	#	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
```

Visualizing a correlation heatmap of the data.

[50]:
```
# 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 the dataset")
plt.show()
```

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, we could exclude `video_like_count`. And among the variables that quantify video metrics, we could keep `video_view_count`, `video_share_count`, `video_download_count`, and `video_comment_count` as features.

2.3 PACE: Construct

2.3.1 3a. Selecting variables

Selecting the outcome variable.

[53]:
```
y = data_upsampled["verified_status"]
```

Selecting the features.

[54]:
```
# 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"]]
# The first few rows of features dataframe
X.head()
```

[54]:

	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

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.

2.3.2 3b. Train-test split

Spliting the data into training and testing sets.

[55]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
↪random_state=0)
```

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

[56]:
```
# shape of each training and testing set
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

[56]:
```
((26826, 7), (8942, 7), (26826,), (8942,))
```

2.3.3 3c. Encoding variables

Checking the data types of the features.

[57]:
```
# Check data types
X_train.dtypes
```

[57]:
```
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
```

[59]:
```
# Get unique values in `claim_status`
X_train['claim_status'].unique()
```

[59]:
```
array(['opinion', 'claim'], dtype=object)
```

[60]:
```
# Get unique values in `author_ban_status`
X_train['author_ban_status'].unique()
```

[60]:
```
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.

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

[61]:
```
# Selecting the training features that needs to be encoded
X_train_to_encode = X_train[["claim_status", "author_ban_status"]]
X_train_to_encode.head()
```

[61]:

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

[63]:
```
# Setting up an encoder for one-hot encoding the categorical features
X_encoder = OneHotEncoder(drop='first', sparse_output=False)
```

[64]:
```
# Fit and transform the training features using the encoder
X_train_encoded = X_encoder.fit_transform(X_train_to_encode)
```

[66]:
```
# Get feature names from encoder
X_encoder.get_feature_names_out()
```

[66]:
```
array(['claim_status_opinion', 'author_ban_status_banned',
       'author_ban_status_under review'], dtype=object)
```

[67]:
```
# Displaying first few rows of encoded training features
X_train_encoded
```

[67]:
```
array([[1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       …,
       [1., 0., 0.],
       [1., 0., 0.],
       [0., 1., 0.]])
```

[68]:
```
# 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())
X_train_encoded_df
```

[68]:

	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	
…	…	…	
26821	1.0	0.0	
26822	1.0	0.0	
26823	1.0	0.0	
26824	1.0	0.0	
26825	0.0	1.0	

	author_ban_status_under review
0	0.0
1	0.0
2	0.0
3	0.0
4	0.0
…	…
26821	0.0
26822	0.0
26823	0.0
26824	0.0
26825	0.0

[26826 rows x 3 columns]

[74]:
```
# Displaying 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()
```

[74]:

	video_duration_sec	video_view_count	video_share_count	video_download_count	video_comment_count
33058	33	2252.0	23.0	4.0	0.0
20491	52	6664.0	550.0	53.0	2.0
25583	37	6327.0	257.0	3.0	0.0
18474	57	1702.0	28.0	0.0	0.0
27312	21	3842.0	101.0	1.0	0.0

[75]:
```
# Concatenate `X_train` and `X_train_encoded_df` to form the final dataframe for training data (`X_train_final`)
X_train_final = pd.concat([X_train.drop(columns=["claim_status", "author_ban_status"]).reset_index(drop=True), X_train_encoded_df], axis=1)

X_train_final.head()
```

[75]:

	video_duration_sec	video_view_count	video_share_count	video_download_count	video_comment_count	claim_status_opinion	author_ban_status_banned	author_ban_status_under review
0	33	2252.0	23.0	4.0	0.0	1.0	0.0	0.0
1	52	6664.0	550.0	53.0	2.0	1.0	0.0	0.0
2	37	6327.0	257.0	3.0	0.0	1.0	0.0	0.0
3	57	1702.0	28.0	0.0	0.0	1.0	0.0	0.0
4	21	3842.0	101.0	1.0	0.0	1.0	0.0	0.0

Checking the data type of the outcome variable.

[77]:
```
y_train.dtype
```

[77]:
```
dtype('O')
```

[79]:
```
# Get unique values of outcome variable
y_train.unique()
```

[79]:
```
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.

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

[80]:
```
# Setting up an encoder for one-hot encoding the categorical outcome variable
y_encoder = OneHotEncoder(drop='first', sparse_output=False)
```

[84]:
```
# Encode the training outcome variable
y_train_final = y_encoder.fit_transform(y_train.values.reshape(-1, 1)).ravel()
y_train_final
```

[84]:
```
array([1., 1., 1., …, 1., 1., 0.])
```

2.3.4 3d. Model building

Constructing a model and fitting it to the training set.

[86]:
```
# 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)
```

2.4 PACE: Execute

2.4.1 4a. Results and evaluation

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

[87]:
```
# Select the testing features that needs to be encoded
X_test_to_encode = X_test[["claim_status", "author_ban_status"]]
X_test_to_encode.head()
```

[87]:

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

[88]:
```
# Transforming the testing features using the encoder
X_test_encoded = X_encoder.transform(X_test_to_encode)
X_test_encoded
```

[88]:
```
array([[1., 0., 0.],
       [1., 0., 0.],
       [0., 0., 0.],
       …,
       [1., 0., 0.],
       [0., 0., 1.],
       [1., 0., 0.]])
```

[89]:
```
# Placing 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())
X_test_encoded_df.head()
```

[89]:

	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

[90]:
```
# Displaying 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()
```

[90]:

	video_duration_sec	video_view_count	video_share_count	video_download_count	video_comment_count
21061	41	2118.0	57.0	5.0	2.0
31748	27	5701.0	157.0	1.0	0.0
20197	31	449767.0	75385.0	5956.0	728.5
5727	19	792813.0	56597.0	5146.0	728.5
11607	54	2044.0	68.0	19.0	2.0

[91]:
```
# Concatenating `X_test` and `X_test_encoded_df` to form the final dataframe␣
↪for training data (`X_test_final`)
X_test_final = pd.concat([X_test.drop(columns=["claim_status",␣
↪"author_ban_status"]).reset_index(drop=True), X_test_encoded_df], axis=1)
X_test_final.head()
```

[91]:

	video_duration_sec	video_view_count	video_share_count	video_download_count	video_comment_count	claim_status_opinion	author_ban_status_banned	author_ban_status_under review
0	41	2118.0	57.0	5.0	2.0	1.0	0.0	0.0
1	27	5701.0	157.0	1.0	0.0	1.0	0.0	0.0
2	31	449767.0	75385.0	5956.0	728.5	0.0	0.0	0.0
3	19	792813.0	56597.0	5146.0	728.5	0.0	0.0	0.0
4	54	2044.0	68.0	19.0	2.0	1.0	0.0	0.0

Testing the logistic regression model. We will use the model to make predictions on the encoded testing set.

[92]:
```
# Use the logistic regression model to get predictions on the encoded testing␣
↪set
y_pred = log_clf.predict(X_test_final)
```

Displaying the predictions on the encoded testing set.

[93]:
```
# Display the predictions on the encoded testing set
y_pred
```

[93]:
```
array([1., 1., 0., …, 1., 0., 1.])
```

Displaying the true labels of the testing set.

[94]:
```
# Display the true labels of the testing set
y_test
```

[94]:
```
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
```

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

[95]:
```
# Encode the testing outcome variable
y_test_final = y_encoder.transform(y_test.values.reshape(-1, 1)).ravel()

# Display the encoded testing outcome variable
y_test_final
```

[95]:
```
array([1., 1., 1., …, 0., 1., 1.])
```

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

[97]:
```
# Get shape of each training and testing set
X_train_final.shape, y_train_final.shape, X_test_final.shape, y_test_final.shape
```

[97]:
```
((26826, 8), (26826,), (8942, 8), (8942,))
```

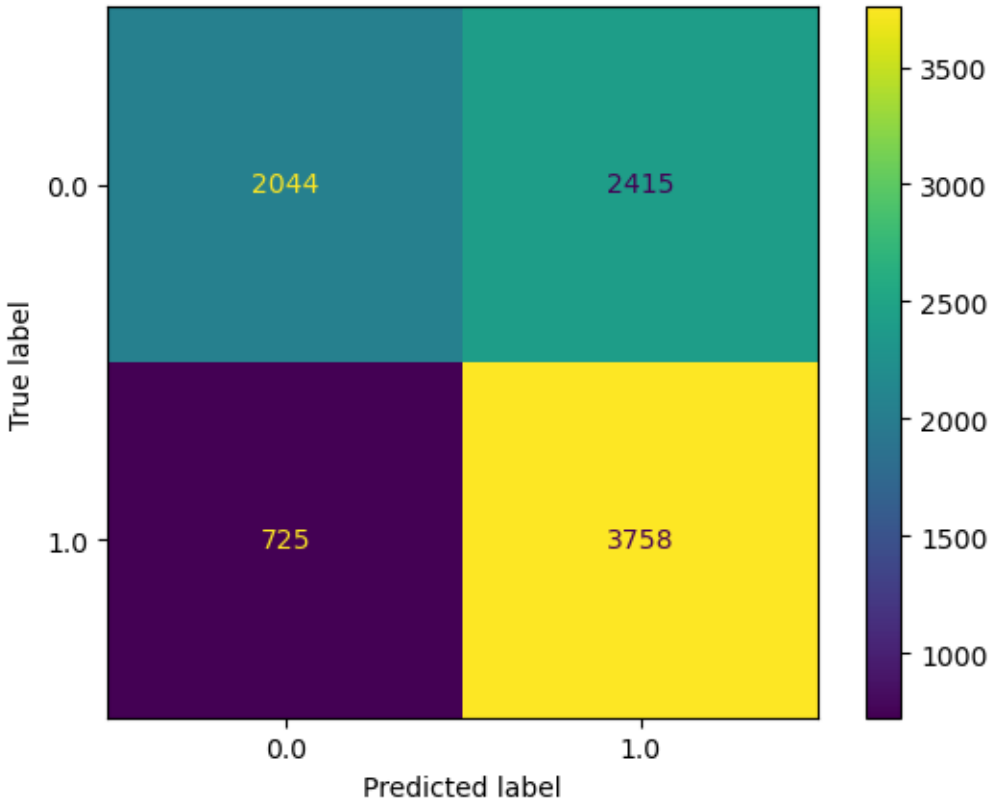
2.4.2 4b. Visualizing model results

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

[98]:
```
log_cm = confusion_matrix(y_test_final, y_pred, labels=log_clf.classes_)

log_disp = ConfusionMatrixDisplay(confusion_matrix=log_cm,␣
 ↪display_labels=log_clf.classes_)

log_disp.plot()
plt.show()
```



[99]:
```
(3758+2044) / (3758 + 725 + 2044 + 2415)
```

[99]:
```
0.6488481324088571
```

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.

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

[100]:
```
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
```

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.

2.4.3 4c. Interpreting model coefficients

[101]:
```
# 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]})
```

[101]:

	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

2.4.4 4d. Conclusion

- 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.