Activity_Course 5 TikTok project lab

August 26, 2023

1 TikTok Project

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

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?

2 Build a regression model

2.0.1 Imports and loading

[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 sb

# 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
import scipy.stats as sp
import sklearn as sk
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
```

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

2.0.2 Explore data with EDA

[3]:
```
# Display first few rows
df.head(5)
```

[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
```

[4]:
```
# Get number of rows and columns
np.shape(df)
```

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

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

[6]:
```
# Get basic information
df.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
```

[7]:
```
# Generate basic descriptive stats
df.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
```

[8]:
```
# Check for missing values
df.isnull().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
df = df.dropna()
df.isnull().sum()
```

[9]:
```
#                         0
claim_status              0
video_id                  0
video_duration_sec        0
video_transcription_text  0
verified_status           0
```

```
author_ban_status           0
video_view_count            0
video_like_count            0
video_share_count           0
video_download_count        0
video_comment_count         0
dtype: int64
```

[10]:
```
# Display first few rows after handling missing values
df.head(5)
```

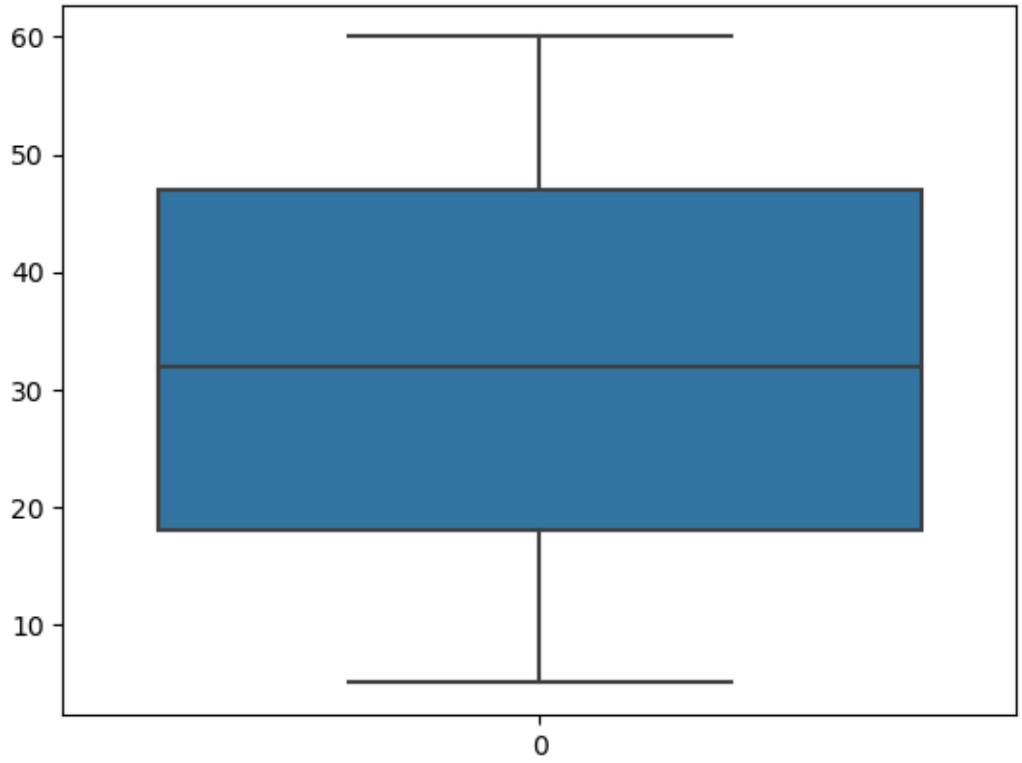
[10]:
```
   # claim_status     video_id  video_duration_sec  \
0  1        claim   7017666017                  59
1  2        claim   4014381136                  32
2  3        claim   9859838091                  31
3  4        claim   1866847991                  25
4  5        claim   7105231098                  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
```

[11]:
```
# Check for duplicates
df.duplicated().sum()
#df.drop_duplicates()
```

[11]:
```
0
```

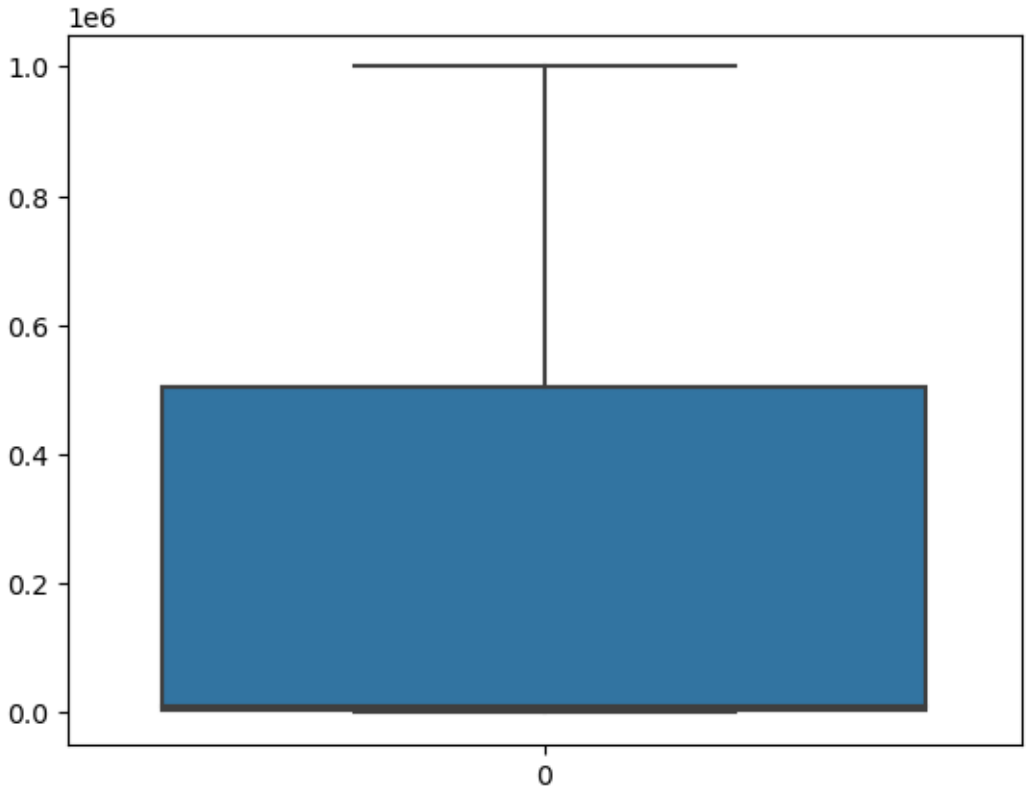
[12]:
```
# Create a boxplot to visualize distribution of `video_duration_sec`
sb.boxplot(df.video_duration_sec)
```

[12]: <Axes: >



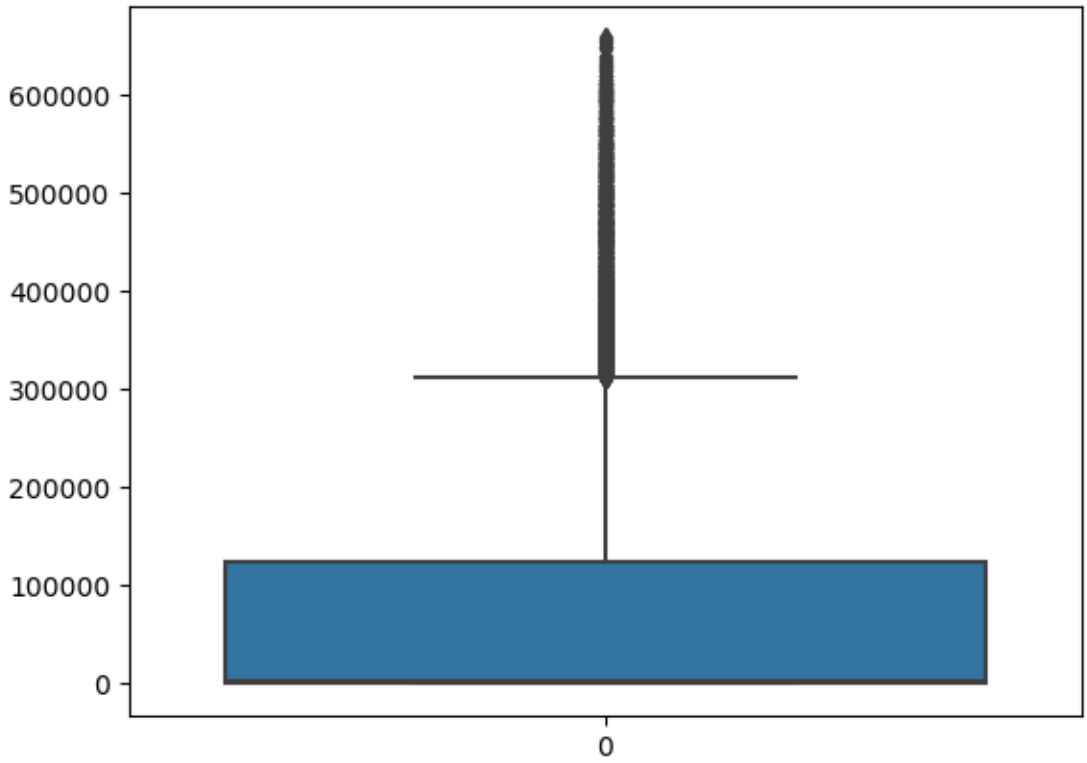
[13]:
```
# Create a boxplot to visualize distribution of `video_view_count`
sb.boxplot(df.video_view_count)
```

[13]: <Axes: >



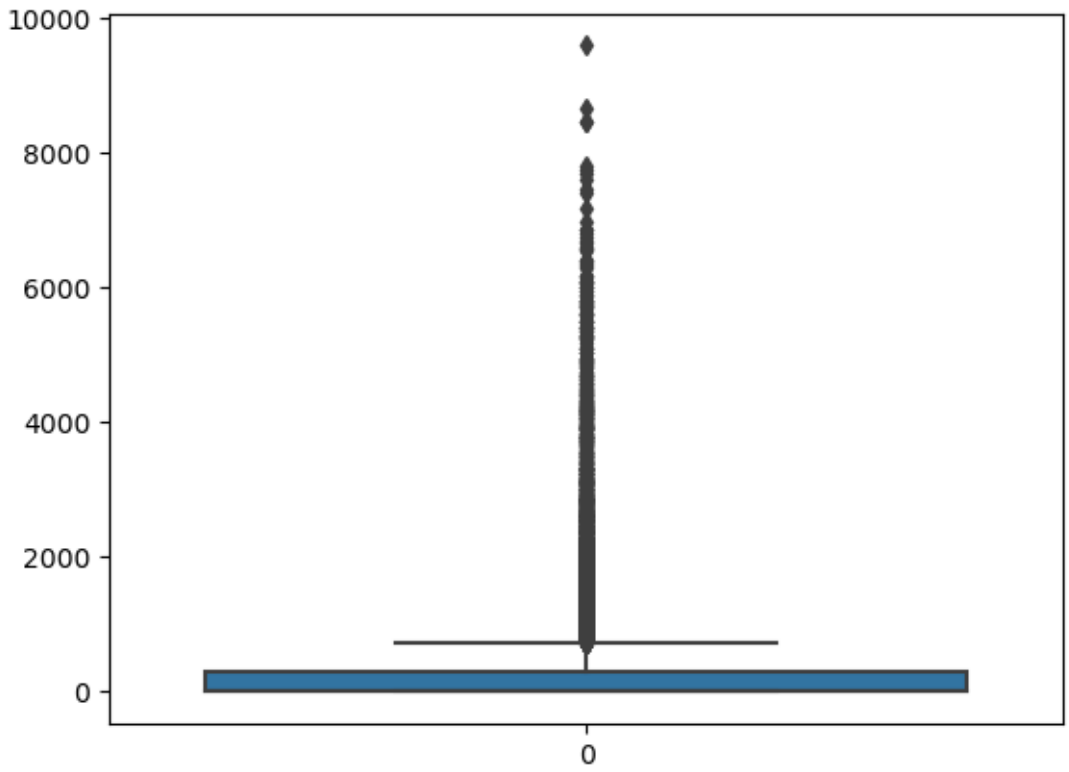
[14]:
```
# Create a boxplot to visualize distribution of `video_like_count`
sb.boxplot(df.video_like_count)
```

[14]: <Axes: >



[15]:
```
# Create a boxplot to visualize distribution of `video_comment_count`
sb.boxplot(df.video_comment_count)
```

[15]: <Axes: >



[16]:
```
# Remove outliers from video_like_count
df.video_like_count.max()
```

[16]: 657830.0

[17]:
```
# Check for and handle outliers for video_like_count
for x in ['video_like_count']:
    q75,q25 = np.nanpercentile(df.loc[:,x],[75,25])
    iqr = q75-q25

    max = np.nanmedian(df.video_like_count)+(1.5*iqr)

    df.loc[df['video_like_count'] > max,'video_like_count'] = max

df.video_like_count.max()
```

[17]: 189717.375

[18]:
```
df.video_comment_count.max()
```

[18]: 9599.0

[19]:
```
# Check for and handle outliers for video_like_count
for x in ['video_comment_count']:
    q75,q25 = np.nanpercentile(df.loc[:,x],[75,25])
    iqr = q75-q25

    max = np.nanmedian(df.video_comment_count)+(1.5*iqr)

    df.loc[df['video_comment_count'] > max,'video_comment_count'] = max

df.video_comment_count.max()
```

[19]:
```
445.5
```

Check class balance.

[20]:
```
# Check class balance for verified_status
df["verified_status"].value_counts(normalize=True)
# verified_status is severely imbalanced.
```

[20]:
```
verified_status
not verified    0.93712
verified        0.06288
Name: proportion, dtype: float64
```

[21]:
```
# Use resampling to create class balance in the outcome variable

# Identify data points from majority and minority classes
Vmask = df[df.verified_status == 'verified']
# Verified (minority class) mask
UVmask = df[df.verified_status == 'not verified']
# Unverified (majority class) mask

print('Verified:',len(Vmask),'Unverified:',len(UVmask),sep="\n")

# Upsample the minority class (which is "verified")
from sklearn.utils import resample
Vmaskup = resample(Vmask,
             replace=True,
             n_samples=len(UVmask),
             random_state=0)
# Verified (minority class) mask upsampled

#print('Verified mask upsampled:',Vmaskup.shape,'Verified upsampled:',np.
 ↪size(Vmaskup),'Unverified:',np.size(UVmask),sep="\n")

# Combine majority class with upsampled minority class
dfup = pd.concat([Vmaskup, UVmask]).reset_index(drop=True)
```

```
#dataframe upsampled

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

```
Verified:
1200
Unverified:
17884
```

[21]:
```
verified_status
verified        17884
not verified    17884
Name: count, dtype: int64
```

[22]:
```
# Get the average `video_transcription_text` length for verified vs not␣
 ↪verified users
#dfup[["verified_status", "video_transcription_text"]].
 ↪groupby(by="verified_status")[["video_transcription_text"]].agg(func=lambda␣
 ↪array: np.mean([len(text) for text in array]))
#dfup['VTL'] = df['video_transcription_text'].str.len()
# This is supposed to work, but it is not currently functioning.

# So I've used this instead:
meanV_VTL = np.nanmean(Vmask['video_transcription_text'].str.len())
meanUV_VTL = np.nanmean(UVmask['video_transcription_text'].str.len())
print('Average video text length for verified users:
 ↪',round(meanV_VTL,1),'Average video text lenth for unverified users:
 ↪',round(meanUV_VTL,1),sep='\n')
```

```
Average video text length for verified users:
84.5
Average video text lenth for unverified users:
89.4
```

[23]:
```
# Extract the length of each `video_transcription_text` and add this as a␣
 ↪column to the dataframe
dfup['VTL'] = dfup['video_transcription_text'].str.len()
```

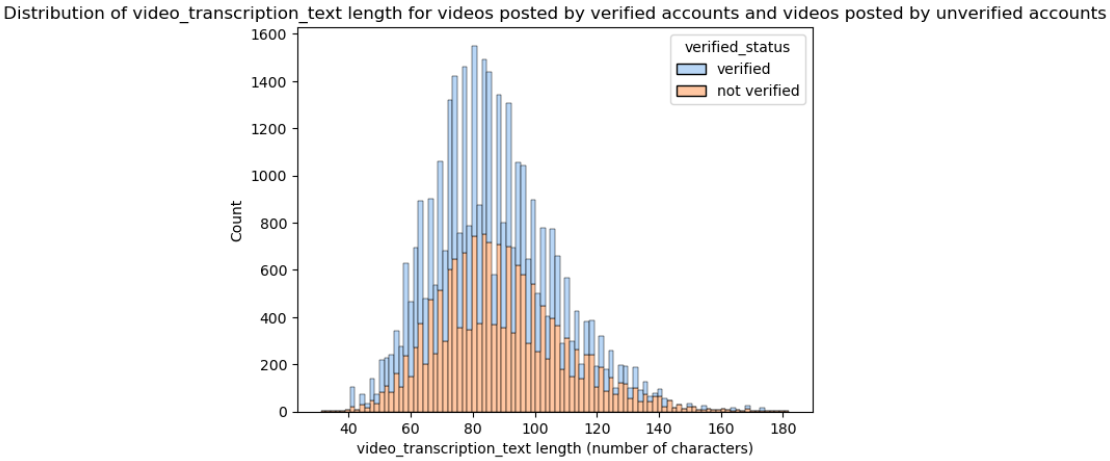
[24]:
```
# Display first few rows of dataframe after adding new column
dfup.head(5)
```

[24]:

	#	claim_status	video_id	video_duration_sec	\
0	14015	opinion	5381182853	58	
1	12805	opinion	2524108154	29	
2	15457	opinion	4462533276	25	
3	14744	opinion	8794309928	24	

```
4  17398      opinion  7566875258                  17

                          video_transcription_text verified_status  \
0  my friends' impression is that a candle’s flam…        verified
1  our impression is that the hummingbird is the …        verified
2  my family's position is that neptune radiates …        verified
3  my family's view is that the longest commercia…        verified
4  i am willing to say that the best selling sing…        verified

  author_ban_status  video_view_count  video_like_count  video_share_count  \
0            active             154.0              36.0                2.0
1            active            9565.0            1505.0              593.0
2            active            9351.0            1850.0               62.0
3            active            7361.0             527.0              207.0
4            active            1528.0              56.0               20.0

   video_download_count  video_comment_count  VTL
0                   1.0                  0.0   71
1                  14.0                  2.0   78
2                  14.0                  2.0   81
3                   1.0                  0.0   86
4                   1.0                  0.0   96
```

[25]:
```
# 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
sb.histplot(data=dfup, stat="count", multiple="stack", x="VTL", 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()
```



2.0.3 Examine correlations

[26]:
```
# Code a correlation matrix to help determine most correlated variables
dfupcorr = dfup.corr(numeric_only=True)
dfupcorr
```

[26]:

	#	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	-0.676775	0.002422	0.004199	
video_share_count	-0.504015	0.010515	0.002206	
video_download_count	-0.487096	0.008753	0.003989	
video_comment_count	-0.656703	0.011955	-0.000321	
VTL	-0.193677	-0.007083	-0.002981	

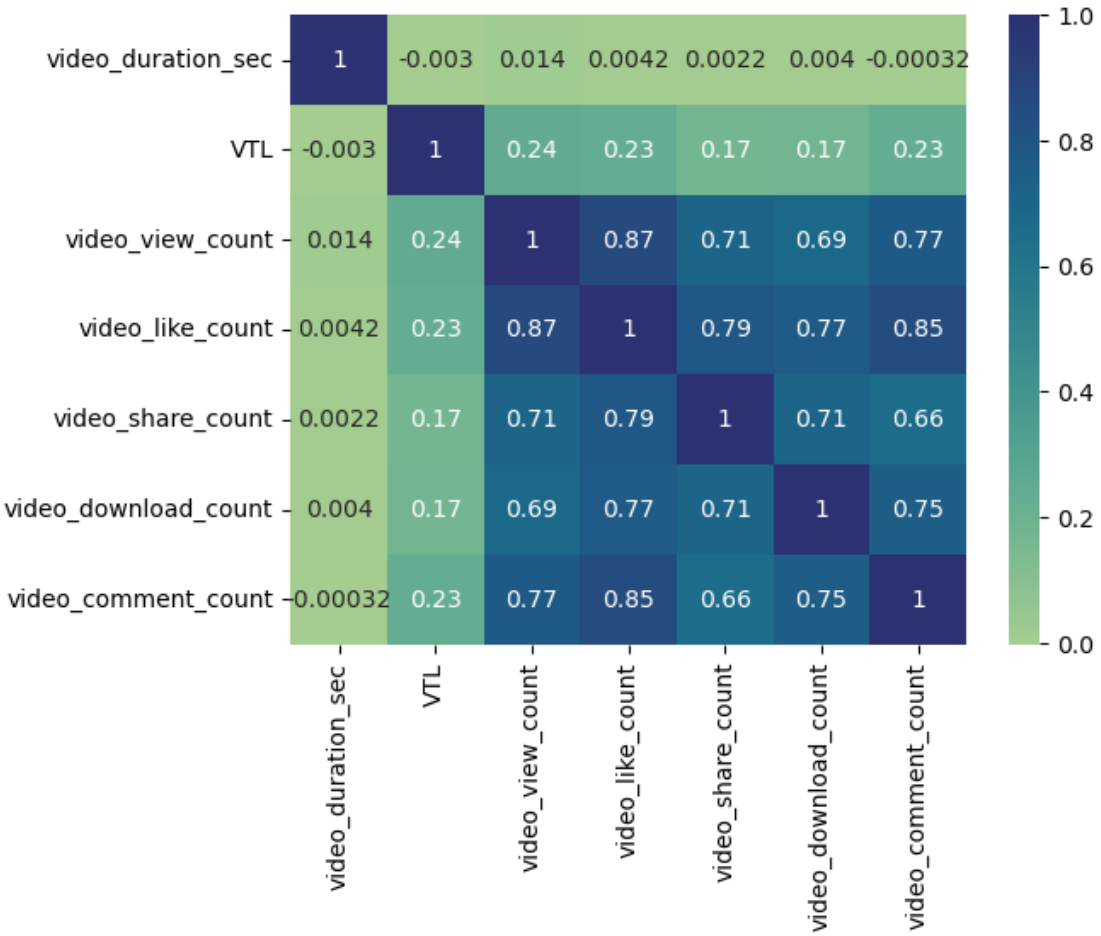
	video_view_count	video_like_count	video_share_count	\
#	-0.697007	-0.676775	-0.504015	
video_id	0.002554	0.002422	0.010515	
video_duration_sec	0.013589	0.004199	0.002206	
video_view_count	1.000000	0.870402	0.711313	
video_like_count	0.870402	1.000000	0.788867	
video_share_count	0.711313	0.788867	1.000000	
video_download_count	0.690048	0.765259	0.710117	
video_comment_count	0.773174	0.852355	0.662942	
VTL	0.244693	0.232931	0.171651	

	video_download_count	video_comment_count	VTL
#	-0.487096	-0.656703	-0.193677
video_id	0.008753	0.011955	-0.007083

```
video_duration_sec          0.003989          -0.000321 -0.002981
video_view_count            0.690048           0.773174  0.244693
video_like_count            0.765259           0.852355  0.232931
video_share_count           0.710117           0.662942  0.171651
video_download_count        1.000000           0.754130  0.173396
video_comment_count         0.754130           1.000000  0.234122
VTL                         0.173396           0.234122  1.000000
```

[27]:
```
# Create a heatmap to visualize how correlated variables are
#plt.figure(figsize=(8, 6))
sb.heatmap(dfup[["video_duration_sec", "VTL", "claim_status",␣
 ↪"author_ban_status", "video_view_count", "video_like_count",␣
 ↪"video_share_count", "video_download_count", "video_comment_count"]].
 ↪corr(numeric_only=True),annot=True, cmap="crest")
```

[27]: <Axes: >



2.0.4 Select variables

[28]:
```
# Select outcome variable
y = dfup[['verified_status']]
```

[29]:
```
# Select features
X =␣
 ↪dfup[['video_duration_sec','VTL','claim_status','author_ban_status','video_view_count','vid

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

[29]:

	video_duration_sec	VTL	claim_status	author_ban_status	video_view_count	\
0	58	71	opinion	active	154.0	
1	29	78	opinion	active	9565.0	
2	25	81	opinion	active	9351.0	
3	24	86	opinion	active	7361.0	
4	17	96	opinion	active	1528.0	

	video_like_count	video_share_count	video_download_count	\
0	36.0	2.0	1.0	
1	1505.0	593.0	14.0	
2	1850.0	62.0	14.0	
3	527.0	207.0	1.0	
4	56.0	20.0	1.0	

	video_comment_count
0	0.0
1	2.0
2	2.0
3	0.0
4	0.0

2.0.5 Train-test split

[30]:
```
# Split the data into training and testing sets
# using the train test split function
X_train, X_test, y_train, y_test = train_test_split(X,y ,
                                    random_state=104,
                                    test_size=0.25,
                                    shuffle=True)
```

[31]:
```
# Get shape of each training and testing set
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

[31]: ((26826, 9), (8942, 9), (26826, 1), (8942, 1))

2.0.6 Encode variables

[32]:
```
# Check data types
X_train.dtypes
```

[32]:
```
video_duration_sec        int64
VTL                       int64
claim_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
```

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

[33]:
```
array(['opinion', 'claim'], dtype=object)
```

[34]:
```
# Get unique values in `author_ban_status`
X_train["author_ban_status"].unique()
```

[34]:
```
array(['active', 'banned', 'under review'], dtype=object)
```

[35]:
```
# 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()
```

[35]:

	claim_status	author_ban_status
3868	opinion	active
8672	opinion	active
35264	opinion	active
18835	claim	active
33949	opinion	active

[36]:
```
# Set up an encoder for one-hot encoding the categorical features
X_encoder = OneHotEncoder(drop='first', sparse_output=False)
```

[37]:
```
# Fit and transform the training features using the encoder
X_train_encoded = X_encoder.fit_transform(X_train_to_encode)
```

[38]:
```
# Get feature names from encoder
X_encoder.get_feature_names_out()
```

[38]:
```
array(['claim_status_opinion', 'author_ban_status_banned',
       'author_ban_status_under review'], dtype=object)
```

[39]:
```
# Display first few rows of encoded training features
X_train_encoded
```

[39]:
```
array([[1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       …,
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.]])
```

[40]:
```
# 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()
```

[40]:

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

[41]:
```
# 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()
```

[41]:

	video_duration_sec	VTL	video_view_count	video_like_count	\
3868	33	71	2252.0	829.0	
8672	16	100	9192.0	3903.0	
35264	5	98	8810.0	1013.0	
18835	7	124	446140.0	188284.0	
33949	38	46	3141.0	981.0	

```
      video_share_count  video_download_count  video_comment_count
3868               23.0                   4.0                  0.0
8672              318.0                  32.0                  6.0
35264             375.0                  11.0                  2.0
18835           38448.0                1702.0                445.5
33949              58.0                  18.0                  1.0
```

[42]:
```
# 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)

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

[42]:
```
   video_duration_sec  VTL  video_view_count  video_like_count  \
0                  33   71            2252.0             829.0
1                  16  100            9192.0            3903.0
2                   5   98            8810.0            1013.0
3                   7  124          446140.0          188284.0
4                  38   46            3141.0             981.0

   video_share_count  video_download_count  video_comment_count  \
0               23.0                   4.0                  0.0
1              318.0                  32.0                  6.0
2              375.0                  11.0                  2.0
3            38448.0                1702.0                445.5
4               58.0                  18.0                  1.0

   claim_status_opinion  author_ban_status_banned  \
0                   1.0                       0.0
1                   1.0                       0.0
2                   1.0                       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
```

[43]:
```
# Check data type of outcome variable
y_train.verified_status.dtype
```

[43]:
```
dtype('O')
```

[44]:
```
# Get unique values of outcome variable
y_train.verified_status.unique()
```

[44]:
```
array(['verified', 'not verified'], dtype=object)
```

[45]:
```
# Set up an encoder for one-hot encoding the categorical outcome variable
y_encoder = OneHotEncoder(drop='first', sparse_output=False)
```

[46]:
```
# Encode the training outcome variable
y_train_final = y_encoder.fit_transform(y_train.values.reshape(-1, 1)).ravel()

# Display the encoded training outcome variable
y_train_final
```

[46]:
```
array([1., 1., 0., …, 1., 0., 1.])
```

2.0.7 Model building

[61]:
```
# 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)
```

2.0.8 Results and evaluation

[48]:
```
# 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()
```

[48]:

	claim_status	author_ban_status
3432	opinion	active
24264	claim	banned
27234	claim	active
14552	opinion	active
32179	opinion	active

[49]:
```
# 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
```

[49]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 0.],
       …,
```

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

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

[50]:

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

[51]:
```
# 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()
```

[51]:

	video_duration_sec	VTL	video_view_count	video_like_count	video_share_count	video_download_count	video_comment_count
3432	25	81	9351.0	1850.000	62.0	14.0	2.0
24264	10	130	430290.0	189717.375	28415.0	3387.0	445.5
27234	49	98	637118.0	157402.000	4842.0	1675.0	445.5
14552	25	99	1535.0	192.000	48.0	1.0	0.0
32179	34	86	5967.0	1695.000	343.0	8.0	0.0

[52]:
```
# Concatenate `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)

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

[52]:

	video_duration_sec	VTL	video_view_count	video_like_count	video_share_count	video_download_count	video_comment_count	claim_status_opinion	author_ban_status_banned	author_ban_status_under review
0	25	81	9351.0	1850.000	62.0	14.0	2.0	1.0	0.0	0.0
1	10	130	430290.0	189717.375	28415.0	3387.0	445.5	0.0	1.0	0.0
2	49	98	637118.0	157402.000	4842.0	1675.0	445.5	0.0	0.0	0.0
3	25	99	1535.0	192.000	48.0	1.0	0.0	1.0	0.0	0.0
4	34	86	5967.0	1695.000	343.0	8.0	0.0	1.0	0.0	0.0

[53]:
```
# Use the logistic regression model to get predictions on the encoded testing␣
  ↪set
y_pred = log_clf.predict(X_test_final)
```

[54]:
```
# Display the predictions on the encoded testing set
y_pred
```

[54]:
```
array([1., 0., 0., …, 1., 1., 1.])
```

[55]:
```
# Display the true labels of the testing set
y_test
```

[55]:
```
      verified_status
3432         verified
24264    not verified
27234    not verified
14552        verified
32179    not verified
…                …
29639    not verified
27760    not verified
8314         verified
2555         verified
30012    not verified

[8942 rows x 1 columns]
```

[56]:
```
# 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
```

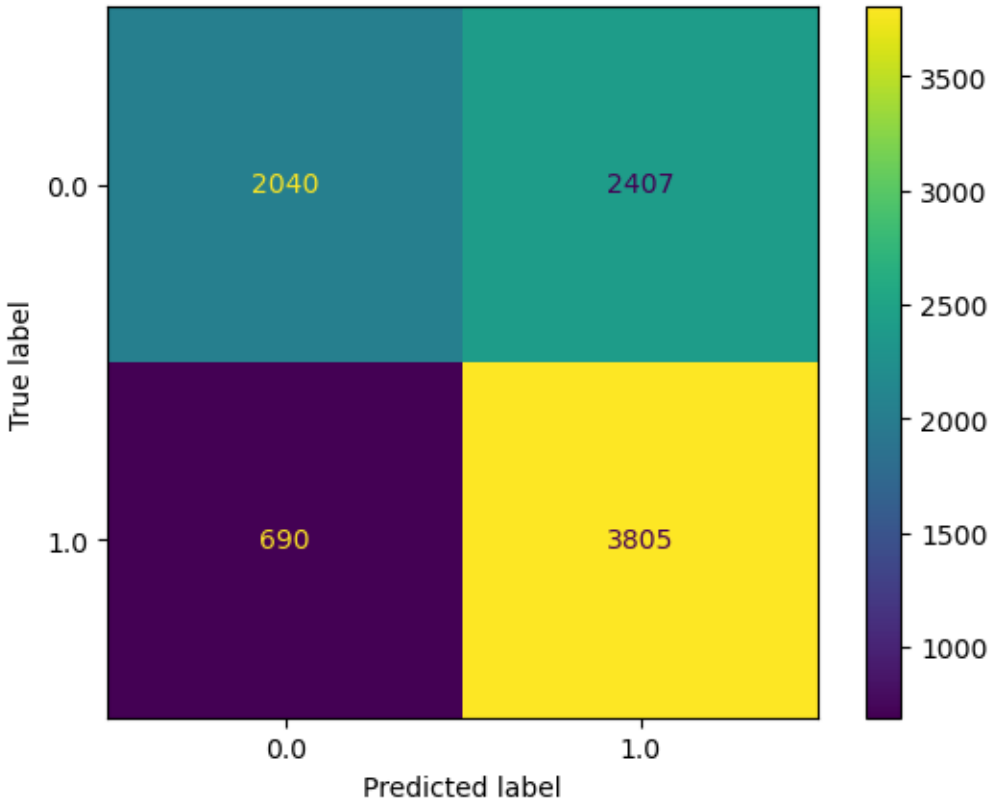
[56]:
```
array([1., 0., 0., …, 1., 1., 0.])
```

[57]:
```
# Get shape of each training and testing set
X_train_final.shape, y_train_final.shape, X_test_final.shape, y_test_final.shape
```

[57]:
```
((26826, 10), (26826,), (8942, 10), (8942,))
```

2.0.9 Visualize model results

[58]:
```
# 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()
```



[59]:
```
# 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       0.75      0.46      0.57      4447
not verified       0.61      0.85      0.71      4495

    accuracy                           0.65      8942
   macro avg       0.68      0.65      0.64      8942
weighted avg       0.68      0.65      0.64      8942
```

2.0.10 Interpret model coefficients

[60]:
```
# 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]})
```

[60]:

	Feature Name	Model Coefficient
0	video_duration_sec	0.001476
1	VTL	0.003771
2	video_view_count	-0.000002
3	video_like_count	-0.000004
4	video_share_count	0.000010
5	video_download_count	0.000010
6	video_comment_count	-0.000964
7	claim_status_opinion	0.000087
8	author_ban_status_banned	-0.000007
9	author_ban_status_under review	-0.000001

2.0.11 Conclusion

1. Key takeaways: The correlation matrix and heatmap show that all of the engagement metrics are correlated somewhat closely. This indicated that it was likely that they might depreciate each other's value in the model. This is confirmed by the fact that the False Positives are quite high. The True Positives and True Negatives are high, but with a high False Positive value as well it is clear that this model is not very realiable. Precision: According to the precision score, accurate Verified predictions are likely to be correct 75% of the time. It's important for the precision in this case to be high since we are trying to make real time predicitions, close to 95% or maybe even 99%, so I would not use this model, but having a decent precision to start with indicates that it might be possible to adjust the variables and find a significantly better model. Recall: This recall value for Not Verified of 85% indicates that this model is already quite good at finding cases that result in Not Verified. It is not very good at finding all of the cases that result in Verified though. F1: The F1 score for Not Verified is over 70%, which is often considered good, but for Verified it is below 70% still, indicating that something needs to be adjusted to improve predictions of Verified accounts.

2. Presentable results: This model is a good first look at how the variables interact, but it is not an acceptable final model. The possible multi-collinearity between the engagement variables is very likely to interfere with the model's accuracy. It's likely that finding out 1 or 2 engagement variables that are the most impactful and only using those may drastically improve the model's accuracy. It would be useful to test forward selection, backward elimination, mixed selection, and all possible models to see if some combination of variables results in a significantly more accurate model that can be used to make good predictions.

[ ]: