Activity_Course 5 TikTok project lab

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

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

4.1.1 Task 1. Imports and loading

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

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

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

# Import packages for data preprocessing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder, OneHotEncoder

# Import packages for data modeling
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.metrics import accuracy_score, recall_score, precision_score,␣
 ↪confusion_matrix

# Specifically for OLS (if you need detailed statistics)
import statsmodels.api as sm

# Setting for better visualization
sns.set_style("whitegrid")
```

```
print("Success")
```

```
Success
```

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

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

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

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

==> ENTER YOUR RESPONSE HERE

4.2.1 Task 2a. Explore data with EDA

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

Inspect the first five rows of the dataframe.

[3]:
```
# Display first few rows

data.head(10)
```

[3]:

	#	claim_status	video_id	video_duration_sec	video_transcription_text	verified_status
0	1	claim	7017666017	59	someone shared with me that drone deliveries a…	not verified
1	2	claim	4014381136	32	someone shared with me that there are more mic…	not verified
2	3	claim	9859838091	31	someone shared with me that american industria…	not verified
3	4	claim	1866847991	25	someone shared with me that the metro of st. p…	not verified
4	5	claim	7105231098	19		
5	6	claim	8972200955	35		
6	7	claim	4958886992	16		
7	8	claim	2270982263	41		
8	9	claim	5235769692	50		
9	10	claim	4660861094	45		

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

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

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

Get the number of rows and columns in the dataset.

[4]:
```
# Get number of rows and columns

data.shape
```

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

Get the data types of the columns.

[5]:
```
# 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]:
```
# Get basic information

print(data.info())

print(data.shape)
```

```
<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
None
(19382, 12)
```

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_isnull = data.isna().sum()

print(data_isnull)
```

```
#                            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_clean = data.dropna()

print(data_clean.shape)
print(data_clean.isna().sum())
```

```
(19084, 12)
#                           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

data_clean.head()
```

[10]:

	#	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				
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				

```
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

duplicates = data_clean.duplicated().sum()

print(f"Total number of duplicates: {duplicates}")
```

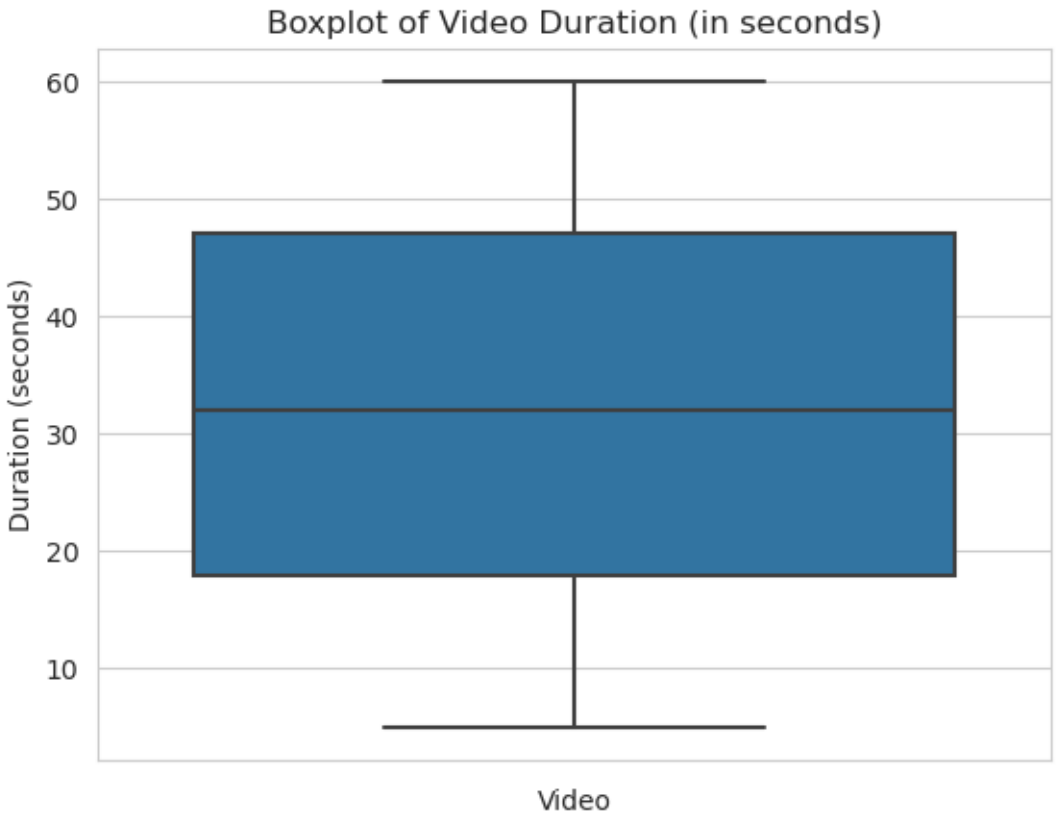
```
Total number of duplicates: 0
```

[12]:
```
df = data_clean.drop_duplicates()

data = df.reset_index(drop=True)

print("^__^")
```

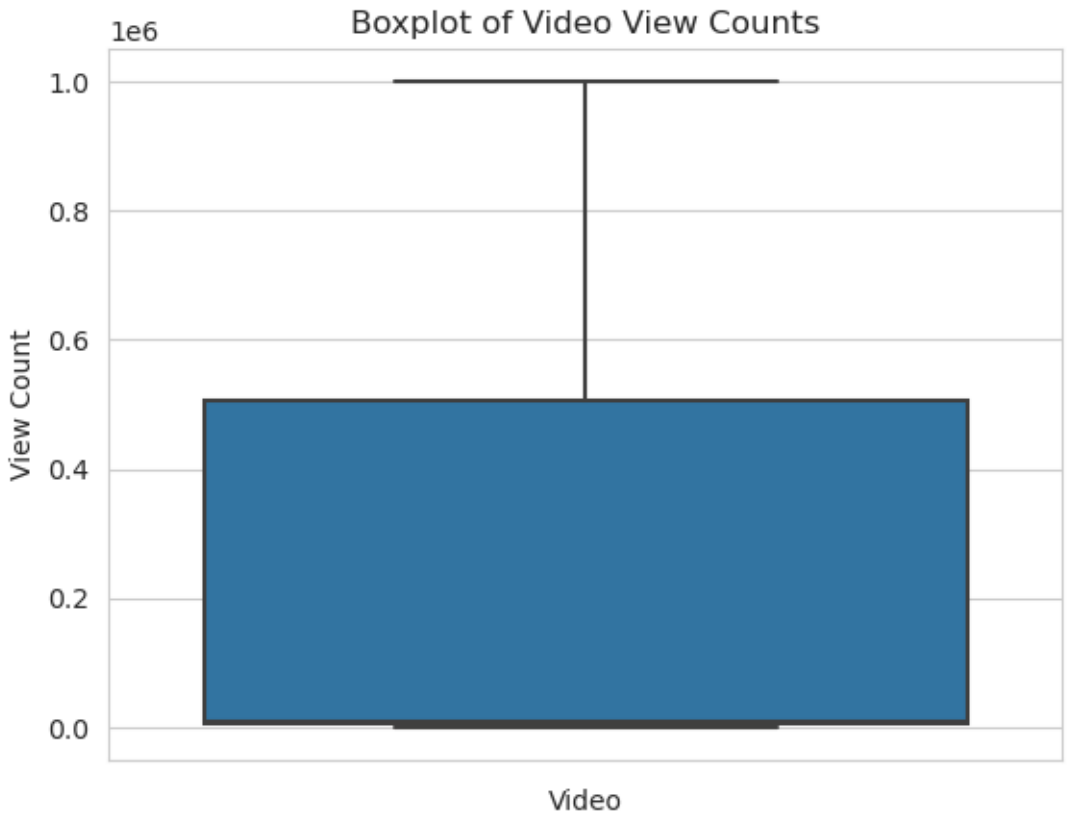
```
^__^
```

Check for and handle outliers.

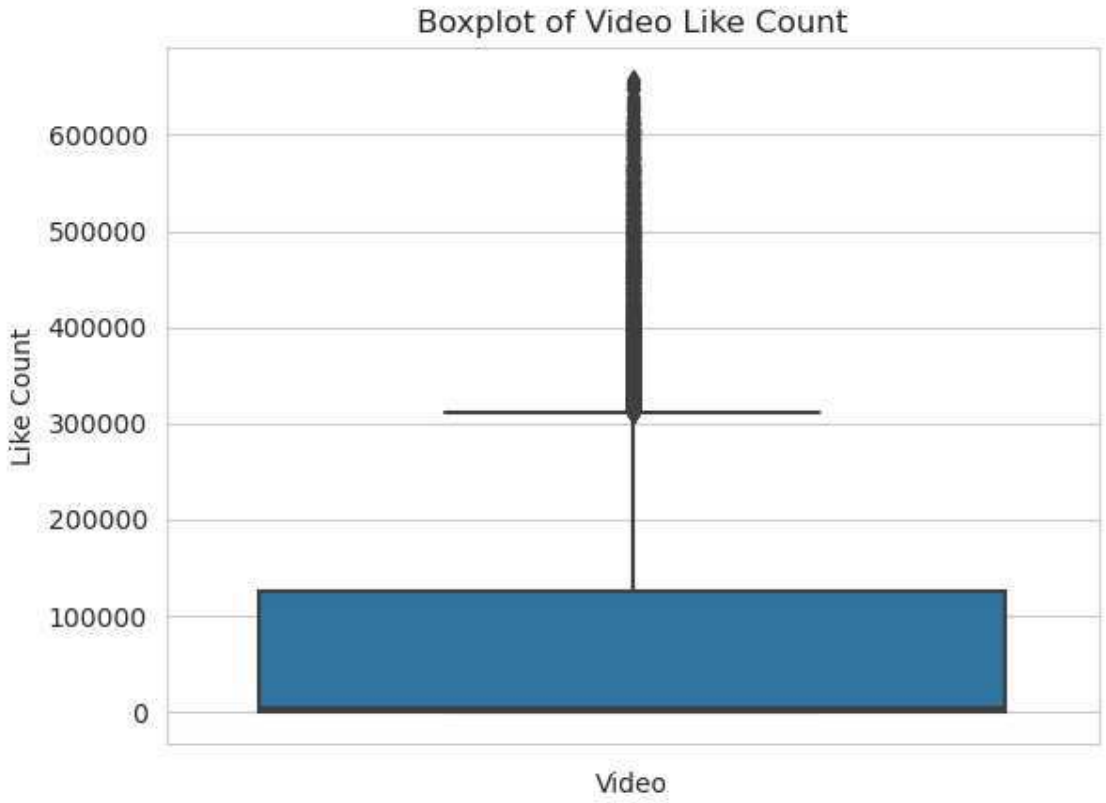
[13]:
```
# Create a boxplot to visualize distribution of `video_duration_sec`

sns.boxplot(y=data['video_duration_sec'])
plt.title('Boxplot of Video Duration (in seconds)')
plt.ylabel('Duration (seconds)')
plt.xlabel('Video')
plt.show()
```



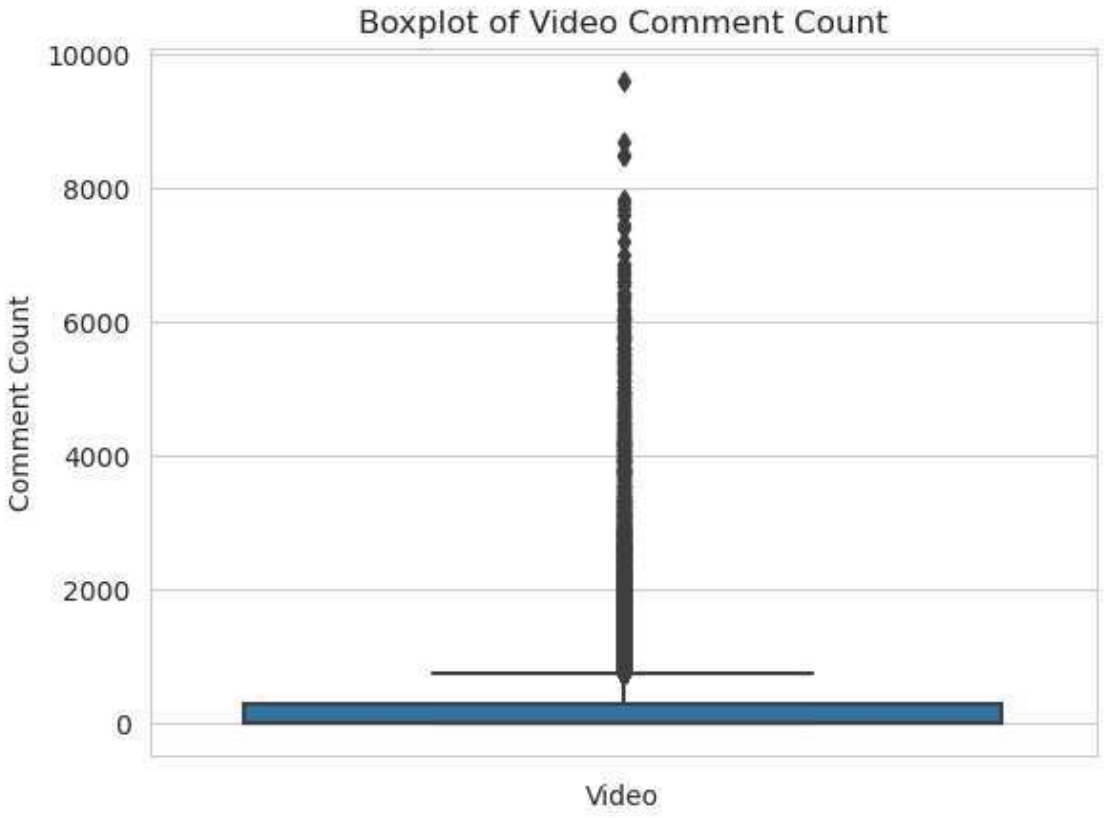
[14]:
```
# Create a boxplot to visualize distribution of `video_view_count`
### YOUR CODE HERE ###

sns.boxplot(y=data['video_view_count'])
plt.title('Boxplot of Video View Counts')
plt.ylabel('View Count')
plt.xlabel('Video')
plt.show()
```



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

sns.boxplot(y=data['video_like_count'])
plt.title('Boxplot of Video Like Count')
plt.ylabel('Like Count')
plt.xlabel('Video')
plt.show()
```



[16]:
```
# Create a boxplot to visualize distribution of `video_comment_count`

sns.boxplot(y=data['video_comment_count'])
plt.title('Boxplot of Video Comment Count')
plt.ylabel('Comment Count')
plt.xlabel('Video')
plt.show()
```



4.3 Capping or Removing outliers

Whether to cap or remove outliers largely depends on the context of the study, the nature of the data, and the goals of the analysis. Since our goal is to build and evaluate a logistic regression model, let's consider the following:

4.3.1 Pros and Cons of Removing Outliers:

Pros: 1. **Improved Model Accuracy**: Outliers can significantly affect the mean and standard deviation of the data, which can lead to misleading results in the logistic regression model. 2. **Better Assumption Fit**: Logistic regression assumes linearity between the logit of the response and the predictors. Outliers can affect this linearity.

Cons: 1. **Loss of Information**: By removing data points, we're losing potential information, and the model might not represent the full variability of data. 2. **Reduction in Sample Size**: If there are many outliers, removing them might result in a significant reduction in the sample size.

4.3.2 Pros and Cons of Capping Outliers:

Pros: 1. **Retain Information**: Instead of discarding outliers, you modify them to fall within a more acceptable range, retaining more of the dataset. 2. **Maintains Sample Size**: You keep the same number of data points.

Cons: 1. **Distortion of Data**: Capping changes the original values, which might lead to distortion of the actual distribution. 2. **Potential Biased Results**: If the capping isn't justified or done correctly, it can introduce bias.

4.3.3 Considerations:

1. **Percentage of Outliers**: If the number of outliers is small, it might be simpler to remove them. If it's a significant portion of our data, capping might be more appropriate.
2. **Nature of Outliers**: If the outliers seem like genuine mistakes or anomalies that aren't representative of any trend or group, it might make sense to remove them. If they represent a specific group or trend, consider keeping or capping them.
3. **Model Purpose**: If the model is exploratory, it might be more flexible. However, if it's for a high-stakes decision, it's important to be more cautious about data manipulation.

4.3.4 Recommendation:

Given the goal is to evaluate the logistic regression model: - **If the number of outliers is small**: Consider removing them to prevent any undue influence on the model's performance. - **If the outliers are significant in number**: Consider capping them to avoid significantly reducing the dataset size.

Additionally, I'd recommend building the model both ways (after capping and after removing) and comparing the performances. This can give you insights into how outliers are affecting the model.

For the sake of this project, I will choose to cap them, but it may produce results different than those provided by the examplar.

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

Q1 = data['video_like_count'].quantile(0.25)
Q3 = data['video_like_count'].quantile(0.75)
IQR = Q3 - Q1

# Define Bounds
lower_bound_like = Q1 - 1.5 * IQR
upper_bound_like = Q3 + 1.5 * IQR

# Handle outliers for video_like_count using loc
data.loc[data['video_like_count'] < lower_bound_like, 'video_like_count'] =␣
 ↪lower_bound_like
data.loc[data['video_like_count'] > upper_bound_like, 'video_like_count'] =␣
 ↪upper_bound_like
```

[18]:
```
# For video_comment_count
Q1_comment = data['video_comment_count'].quantile(0.25)
Q3_comment = data['video_comment_count'].quantile(0.75)
IQR_comment = Q3_comment - Q1_comment

# Define Bounds
```

```
lower_bound_comment = Q1_comment - 1.5 * IQR_comment
upper_bound_comment = Q3_comment + 1.5 * IQR_comment

# Handle outliers for video_comment_count using loc
data.loc[data['video_comment_count'] < lower_bound_comment,␣
 ↪'video_comment_count'] = lower_bound_comment
data.loc[data['video_comment_count'] > upper_bound_comment,␣
 ↪'video_comment_count'] = upper_bound_comment
```

[19]:
```
# Count Outliers to decide whether to cap or remove variables

# Define Bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

num_outliers = ((data['video_like_count'] < lower_bound) |␣
 ↪(data['video_like_count'] > upper_bound)).sum()

print(f"Number of outliers: {num_outliers}")
```

```
Number of outliers: 0
```

[20]:
```
# Count Outliers to decide whether to cap or remove variables

num_outliers_comment = ((data['video_comment_count'] < lower_bound) |␣
 ↪(data['video_comment_count'] > upper_bound)).sum()

print(f"Number of outliers: {num_outliers_comment}")
```

```
Number of outliers: 0
```

Given the context, let's first assess the proportion of outliers in relation to the dataset:

{Proportion of Outliers} = {Number of Outliers}/{Total Data Points}

{Proportion of Outliers} = {1726}{19084} = approx 0.0905

This means that about 9.05% of the data is considered an outlier with respect to the `video_like_count` variable.

Decision to Cap or Remove:

1. **Capping (Winsorizing)**:
   - **Pros**: we retain all data points, preserving data volume and potentially important information from other variables for these records.
   - **Cons**: we extreme values get replaced with more "acceptable" extreme values (the bounds), which might lead to some distortion in the data distribution.
2. **Removing**:
   - **Pros**: The model won't be influenced by extreme values. If these outliers are errors or represent a special case not generalizable to other data, removal can be beneficial.

- **Cons**: You lose about 9% of your dataset, which could also mean losing other valuable information in the other 11 columns for those records. Plus, if these outliers are legitimate observations, you're ignoring potentially valuable data patterns.

Recommendation:

Given that: - The outliers constitute about 9% of the data, - we aim to build a logistic regression model (which can be sensitive to outliers),

I would initially recommend capping (winsorizing) the outliers. This way, we retain the records, and the effects of the outliers will be mitigated.

However, a definitive decision often requires deeper domain knowledge and sometimes even iterative model building:

1. we could try building the model with and without capping (and even removing) and compare the model's performance.
2. If the model's goal relates directly to the `video_like_count`, consider the importance of extreme values for the target audience of the model. If outliers represent unique, important cases, they should potentially be studied separately.

Check class balance.

[21]:
```
# Using pandas to check the count of each category
count_balance = data['verified_status'].value_counts()

print(count_balance)
```

```
verified_status
not verified    17884
verified         1200
Name: count, dtype: int64
```

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.

[22]:
```
# Visualizing the categorical data
sns.countplot(data=data, x='verified_status')
plt.title('Distribution of Verified Status')
plt.show()
```

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

[23]:
```
# Use resampling to create class balance in the outcome variable, if needed

from sklearn.utils import resample

# 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
minority_upsampled = resample(minority,
                              replace=True,       # Sample with replacement
                              n_samples=len(majority),  # Match number in␣
 ↪majority class
                              random_state=42)    # Reproducible results

# Combine majority class with upsampled minority class
data_balanced = pd.concat([majority, minority_upsampled])
```

```
# Display new class counts
print(data_balanced.verified_status.value_counts())
```

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

After executing this code, both classes (verified and not verified) should have equal counts in df_upsampled. This technique creates duplicates of the verified class to balance the classes. Note that while this approach addresses the imbalance, it might make the model more prone to overfitting to the verified class due to the duplication of records. Always validate your model using out-of-sample data to ensure its generalization ability.

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.

[24]:
```
# Get the average `video_transcription_text` length for claims and the average␣
↪`video_transcription_text` length for opinions

# Calculate the average length of video_transcription_text
avg_length = data['video_transcription_text'].apply(len).mean()

print(f"Average transcription length: {avg_length}")
```

```
Average transcription length: 89.09353385034584
```

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.

[25]:
```
# Extract the length of each `video_transcription_text` and add this as a␣
↪column to the dataframe

# Extract the length of each video_transcription_text and add it to a new column
data['transcription_length'] = data['video_transcription_text'].apply(len)


## character length
```

[26]:
```
# Display first few rows of dataframe after adding new column

data.head()
```

[26]:

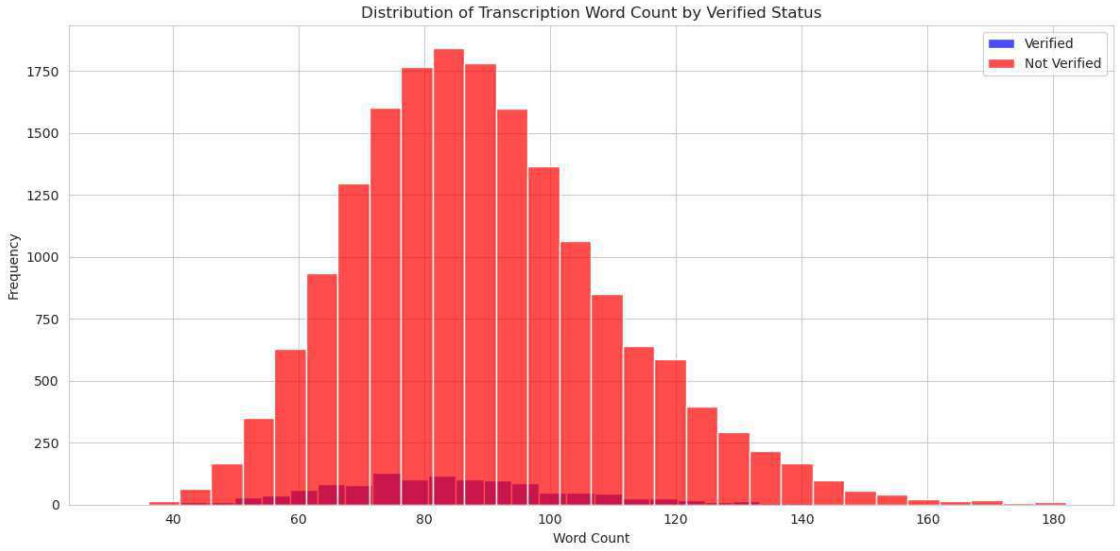
	#	claim_status	video_id	video_duration_sec	\
0	1	claim	7017666017	59	
1	2	claim	4014381136	32	
2	3	claim	9859838091	31	
3	4	claim	1866847991	25	
4	5	claim	7105231098	19	

```
                          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  transcription_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.

[27]:
```
# 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

# Set up the figure and axes
plt.figure(figsize=(12, 6))

# Plot histogram for verified accounts
sns.histplot(data[data['verified_status'] ==␣
 ↪'verified']['transcription_length'], color='blue', kde=False,␣
 ↪label='Verified', bins=30, alpha=0.7)

# Plot histogram for unverified accounts
sns.histplot(data[data['verified_status'] == 'not␣
 ↪verified']['transcription_length'], color='red', kde=False, label='Not␣
 ↪Verified', bins=30, alpha=0.7)

# Set plot title, labels,and legend
plt.title('Distribution of Transcription Word Count by Verified Status')
plt.xlabel('Word Count')
plt.ylabel('Frequency')
plt.legend()
```

```
plt.tight_layout()

#display the plot
plt.show()
```



4.3.5 Task 2b. Examine correlations

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

[28]:
```
# Code a correlation matrix to help determine most correlated variables
data.corr(numeric_only=True)
```

[28]:

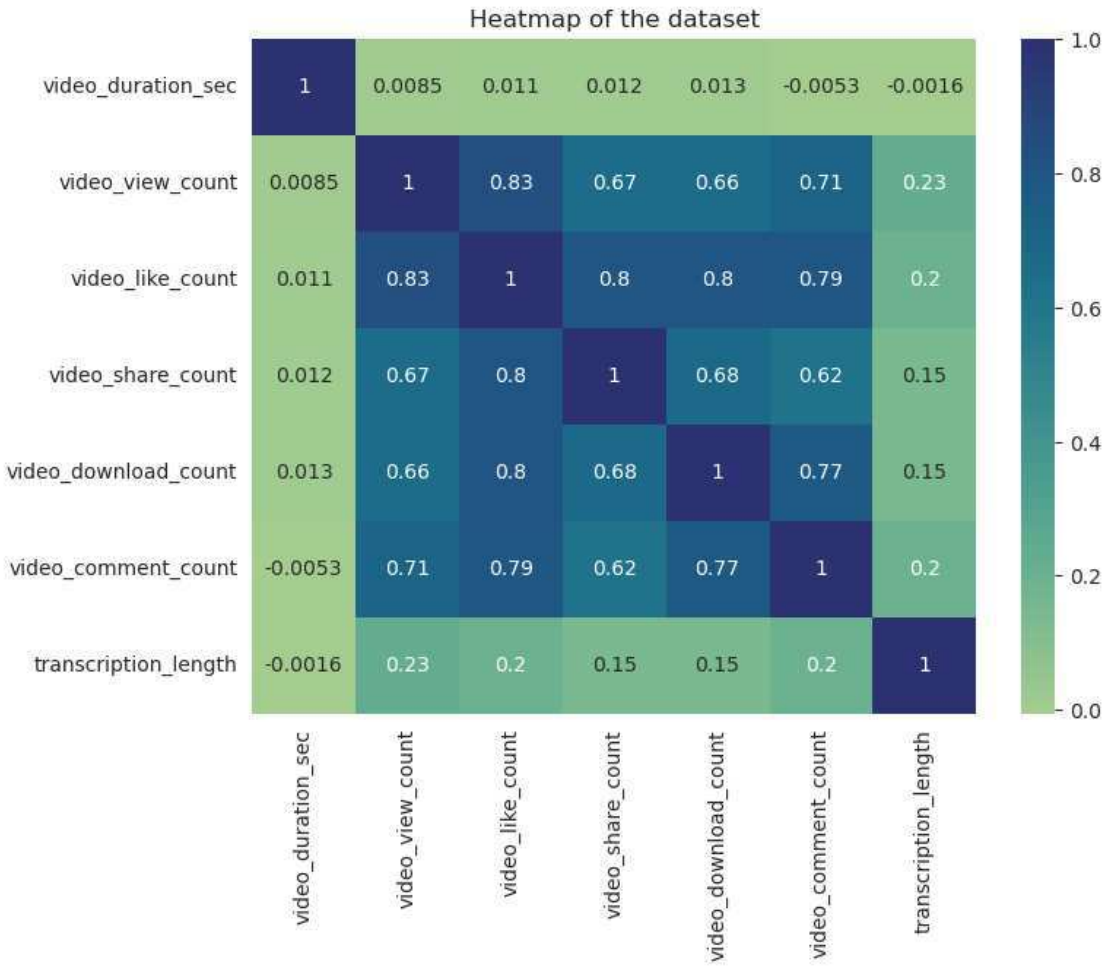
	#	video_id	video_duration_sec	\
#	1.000000	-0.003590	-0.000145	
video_id	-0.003590	1.000000	0.008460	
video_duration_sec	-0.000145	0.008460	1.000000	
video_view_count	-0.668047	0.000217	0.008481	
video_like_count	-0.592271	0.000699	0.010683	
video_share_count	-0.451713	-0.002721	0.011560	
video_download_count	-0.447729	0.002155	0.013078	
video_comment_count	-0.567260	0.004389	-0.005260	
transcription_length	-0.226844	0.000723	-0.001580	

	video_view_count	video_like_count	video_share_count	\
#	-0.668047	-0.592271	-0.451713	
video_id	0.000217	0.000699	-0.002721	
video_duration_sec	0.008481	0.010683	0.011560	
video_view_count	1.000000	0.829156	0.665635	
video_like_count	0.829156	1.000000	0.799829	

```
video_share_count             0.665635          0.799829          1.000000
video_download_count          0.664222          0.797915          0.679910
video_comment_count           0.705983          0.789126          0.620466
transcription_length          0.230212          0.195439          0.147223

                      video_download_count  video_comment_count  \
#                                -0.447729            -0.567260
video_id                          0.002155             0.004389
video_duration_sec                0.013078            -0.005260
video_view_count                  0.664222             0.705983
video_like_count                  0.797915             0.789126
video_share_count                 0.679910             0.620466
video_download_count              1.000000             0.768873
video_comment_count               0.768873             1.000000
transcription_length              0.146382             0.196472

                      transcription_length
#                                -0.226844
video_id                          0.000723
video_duration_sec               -0.001580
video_view_count                  0.230212
video_like_count                  0.195439
video_share_count                 0.147223
video_download_count              0.146382
video_comment_count               0.196472
transcription_length              1.000000
```

Visualize a correlation heatmap of the data.

[29]:
```
# Create a heatmap to visualize how correlated variables are
plt.figure(figsize=(8, 6))
sns.heatmap(
    data[["video_duration_sec", "claim_status", "author_ban_status",␣
 ↪"video_view_count",
                   "video_like_count", "video_share_count",␣
 ↪"video_download_count", "video_comment_count", "transcription_length"]]
    .corr(numeric_only=True),
    annot=True,
    cmap="crest")
plt.title("Heatmap of the dataset")
plt.show()
```



[30]:
```
# Compute correlation matrix for numeric columns
numeric_data = data.select_dtypes(include=['float64', 'int64'])
correlation_matrix = numeric_data.corr()

# Store the correlation matrix in a DataFrame
df_correlation = pd.DataFrame(correlation_matrix)

# Resetting index of the correlation matrix to melt it
correlation_melted = df_correlation.reset_index().melt(id_vars='index')

# Renaming columns for clarity
correlation_melted.columns = ['Variable_1', 'Variable_2', 'Correlation']

print(correlation_melted.head(80))
```

```
    Variable_1  Variable_2  Correlation
0            #           #     1.000000
```

```
1               video_id                      #    -0.003590
2     video_duration_sec                      #    -0.000145
3       video_view_count                      #    -0.668047
4       video_like_count                      #    -0.592271
..                    …                      …          …
75      video_view_count  transcription_length     0.230212
76      video_like_count  transcription_length     0.195439
77     video_share_count  transcription_length     0.147223
78  video_download_count  transcription_length     0.146382
79   video_comment_count  transcription_length     0.196472

[80 rows x 3 columns]
```

The data above indicates pairwise correlations between variables.

For example, the correlation between the column labeled `#` and itself is 1, as expected. Any variable is always perfectly correlated with itself.

Further down, we can see the correlation between `video_view_count` and `transcription_length` is about 0.230212. This means there is a weak positive correlation between the length of the video transcription and the number of views the video gets. Similarly, other variables show their relationship in terms of correlation.

Here's how you can interpret the correlation values:

1. A correlation close to 1 implies a strong positive correlation: as one variable increases, the other also tends to increase.
2. A correlation close to -1 implies a strong negative correlation: as one variable increases, the other tends to decrease.
3. A correlation close to 0 implies little to no relationship between the variables.

With this understanding, we can analyze the relationships between different video metrics and features. For instance, if we find a strong positive correlation between `video_like_count` and `video_view_count`, it could suggest that videos with more views tend to get more likes–it signifies a relationship.

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?

4.4 PACE: Construct

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

4.4.1 Task 3a. Select variables

Set your Y and X variables.

Select the outcome variable.

[31]:
```
# Select outcome variable

y = data['verified_status']
```

Select the features.

[32]:
```
# Select features
X = data[["video_duration_sec", "claim_status", "author_ban_status",␣
 ↪"video_view_count", "video_share_count", "video_download_count",␣
 ↪"video_comment_count"]]

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

[32]:

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

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

4.4.2 Task 3b. Train-test split

Split the data into training and testing sets.

[33]:
```
# Split the data into training and testing sets

# Splitting the data into 80% training and 20% testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
 ↪random_state=0)
```

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

[34]:
```
# Display shape of each set
print(f"X_train shape: {X_train.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"X_test shape: {X_test.shape}")
print(f"y_test shape: {y_test.shape}")
```

```
X_train shape: (14313, 7)
y_train shape: (14313,)
```

```
X_test shape: (4771, 7)
y_test shape: (4771,)
```

4.4.3 Task 3c. Encode variables

Check the data types of the features.

[35]:
```
# Check data types
print(data.dtypes)
```

```
#                             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
transcription_length          int64
dtype: object
```

[36]:
```
# Get unique values in `claim_status`
print("\nUnique values in 'claim_status':")
print(data['claim_status'].unique())
```

```
Unique values in 'claim_status':
['claim' 'opinion']
```

[37]:
```
# Get unique values in `author_ban_status`
print("\nUnique values in 'author_ban_status':")
print(data['author_ban_status'].unique())
```

```
Unique values in 'author_ban_status':
['under review' 'active' 'banned']
```

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.

[38]:
```
# 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()
```

[38]:

	claim_status	author_ban_status
8040	claim	under review
18312	opinion	active
7349	claim	active
2856	claim	active
16585	opinion	under review

[39]:
```
from sklearn.preprocessing import OneHotEncoder

# Set up an encoder for one-hot encoding the categorical features
encoder = OneHotEncoder(drop='first')
```

[40]:
```
# Fit and transform the training features using the encoder
X_train_encoded = encoder.fit_transform(X_train)
```

[41]:
```
# Get feature names from encoder
feature_names = encoder.get_feature_names_out(input_features=X_train.columns)
```

[42]:
```
# Display first few rows of encoded training features
print(X_train_encoded[:5])
```

```
  (0, 28)       1.0
  (0, 57)       1.0
  (0, 10991)    1.0
  (0, 13465)    1.0
  (0, 19751)    1.0
  (0, 23296)    1.0
  (1, 34)       1.0
  (1, 55)       1.0
  (1, 554)      1.0
  (1, 12332)    1.0
  (2, 48)       1.0
  (2, 7398)     1.0
  (2, 16725)    1.0
  (2, 22322)    1.0
  (2, 23978)    1.0
  (3, 40)       1.0
  (3, 9406)     1.0
  (3, 18629)    1.0
  (3, 21751)    1.0
  (3, 23717)    1.0
  (4, 34)       1.0
  (4, 55)       1.0
  (4, 57)       1.0
```

```
  (4, 1442)     1.0
  (4, 12547)    1.0
```

[43]:
```
# Place encoded training features (which is currently an array) into a dataframe

X_train_encoded_df= pd.DataFrame(X_train_encoded.toarray(),␣
 ↪columns=feature_names)


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

```
   video_duration_sec_6  video_duration_sec_7  video_duration_sec_8  \
0                   0.0                   0.0                   0.0
1                   0.0                   0.0                   0.0
2                   0.0                   0.0                   0.0
3                   0.0                   0.0                   0.0
4                   0.0                   0.0                   0.0

   video_duration_sec_9  video_duration_sec_10  video_duration_sec_11  \
0                   0.0                    0.0                    0.0
1                   0.0                    0.0                    0.0
2                   0.0                    0.0                    0.0
3                   0.0                    0.0                    0.0
4                   0.0                    0.0                    0.0

   video_duration_sec_12  video_duration_sec_13  video_duration_sec_14  \
0                    0.0                    0.0                    0.0
1                    0.0                    0.0                    0.0
2                    0.0                    0.0                    0.0
3                    0.0                    0.0                    0.0
4                    0.0                    0.0                    0.0

   video_duration_sec_15  …  video_comment_count_720.0  \
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

   video_comment_count_721.0  video_comment_count_722.0  \
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

   video_comment_count_723.0  video_comment_count_724.0  \
```

```
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

   video_comment_count_725.0  video_comment_count_726.0  \
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

   video_comment_count_727.0  video_comment_count_728.0  \
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

   video_comment_count_728.5
0                         0.0
1                         0.0
2                         1.0
3                         0.0
4                         0.0

[5 rows x 23979 columns]
```

[44]:
```
# 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_dropped = X_train.drop(['claim_status', 'author_ban_status'], axis=1)
print(X_train_dropped.head())
```

```
       video_duration_sec  video_view_count  video_share_count  \
8040                   34          816695.0             1482.0
18312                  40             903.0                8.0
7349                   54          311925.0            27014.0
2856                   46          598217.0            75662.0
16585                  40            2660.0              223.0

       video_download_count  video_comment_count
8040                  185.0                 24.0
18312                   0.0                  0.0
7349                 4427.0                728.5
2856                 2947.0                448.0
16585                   0.0                  0.0
```

[45]:
```
# 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_dropped.reset_index(drop=True),␣
 ↪X_train_encoded_df], axis=1)


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

```
   video_duration_sec  video_view_count  video_share_count  \
0                  34          816695.0             1482.0
1                  40             903.0                8.0
2                  54          311925.0            27014.0
3                  46          598217.0            75662.0
4                  40            2660.0              223.0

   video_download_count  video_comment_count  video_duration_sec_6  \
0                 185.0                 24.0                   0.0
1                   0.0                  0.0                   0.0
2                4427.0                728.5                   0.0
3                2947.0                448.0                   0.0
4                   0.0                  0.0                   0.0

   video_duration_sec_7  video_duration_sec_8  video_duration_sec_9  \
0                   0.0                   0.0                   0.0
1                   0.0                   0.0                   0.0
2                   0.0                   0.0                   0.0
3                   0.0                   0.0                   0.0
4                   0.0                   0.0                   0.0

   video_duration_sec_10  …  video_comment_count_720.0  \
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

   video_comment_count_721.0  video_comment_count_722.0  \
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
```

```
   video_comment_count_723.0  video_comment_count_724.0  \
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

   video_comment_count_725.0  video_comment_count_726.0  \
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

   video_comment_count_727.0  video_comment_count_728.0  \
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

   video_comment_count_728.5
0                        0.0
1                        0.0
2                        1.0
3                        0.0
4                        0.0

[5 rows x 23984 columns]
```

Check the data type of the outcome variable.

[46]:
```
# Check data type of outcome variable
print(y_train.dtype)
```

```
object
```

[47]:
```
# Get unique values of outcome variable
print(y_train.unique())
```

```
['not verified' 'verified']
```

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.

[48]:
```
# Set up an encoder for one-hot encoding the categorical outcome variable
encoder_y = OneHotEncoder(sparse=False)
```

[49]:
```
# 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 = encoder_y.fit_transform(y_train.values.reshape(-1, 1)).ravel()

# Display the encoded training outcome variable
print(y_train_final)
```

```
[1. 0. 0. … 0. 1. 0.]
```

```
/opt/conda/lib/python3.11/site-packages/sklearn/preprocessing/_encoders.py:972:
FutureWarning: `sparse` was renamed to `sparse_output` in version 1.2 and will
be removed in 1.4. `sparse_output` is ignored unless you leave `sparse` to its
default value.
  warnings.warn(
```

4.4.4 Task 3d. Model building

Construct a model and fit it to the training set.

[ ]:
```
# 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)
```

4.5 PACE: Execute

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

4.5.1 Taks 4a. Results and evaluation

Evaluate your model.

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

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

[ ]:
```
# Transform the testing features using the encoder

X_test_encoded = encoder_X.transform(X_test_to_encode)

# Display first few rows of encoded testing features
X_test_encoded.head()
```

[ ]:
```
# Place encoded testing features (which is currently an array) into a dataframe

X_test_encoded_df = pd.DataFrame(X_test_encoded, columns=encoder_X.
 ↪get_feature_names_out(X_test_to_encode.columns))

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

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

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

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

[ ]:
```
# 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.

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

Display the true labels of the testing set.

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

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

```
[ ]: # 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
```

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

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

4.5.2 Task 4b. Visualize model results

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

```
[ ]: # 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.

```
[ ]: # Create a classification report
target_labels = ["verified", "not verified"]
print(classification_report(y_test_final, y_pred, target_names=target_labels))
```

4.5.3 Task 4c. Interpret model coefficients

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

4.5.4 Task 4d. Conclusion

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

4.5.5 Main Insights:

- About the Data Relationships: In our data, we noticed that some pieces of information move together very closely. Think of this as noticing that when it often rains, people tend to carry umbrellas. This is useful, but when these relations are too strong, it can make it tricky for us to pinpoint the exact reason behind a certain outcome. To avoid confusion, we decided to leave out the "number of likes a video gets" from our study.
- Video Length Matters: Our findings suggest that the longer a video is, the higher the chance that it's from a verified user. Imagine it this way: longer videos often come from more established or serious users, much like a well-researched documentary might come from a recognized filmmaker.
- Model Performance in Simple Terms: Our model to predict whether a user is verified based on video features works decently. It's like trying to guess if a house is expensive based on its size and location. We get it right a good amount of the time, but we're not perfect. We're especially good at identifying true verified users, but we sometimes think a user is verified when they're not.
- Concluding Thoughts: We built a tool that tries to guess if a TikTok user is verified by looking at certain features of their videos, like how long they are. This tool does a decent job. Longer videos seem to be a good sign that a user might be verified. However, other features, like how often a video is shared or downloaded, don't tell us as much as we'd hope.