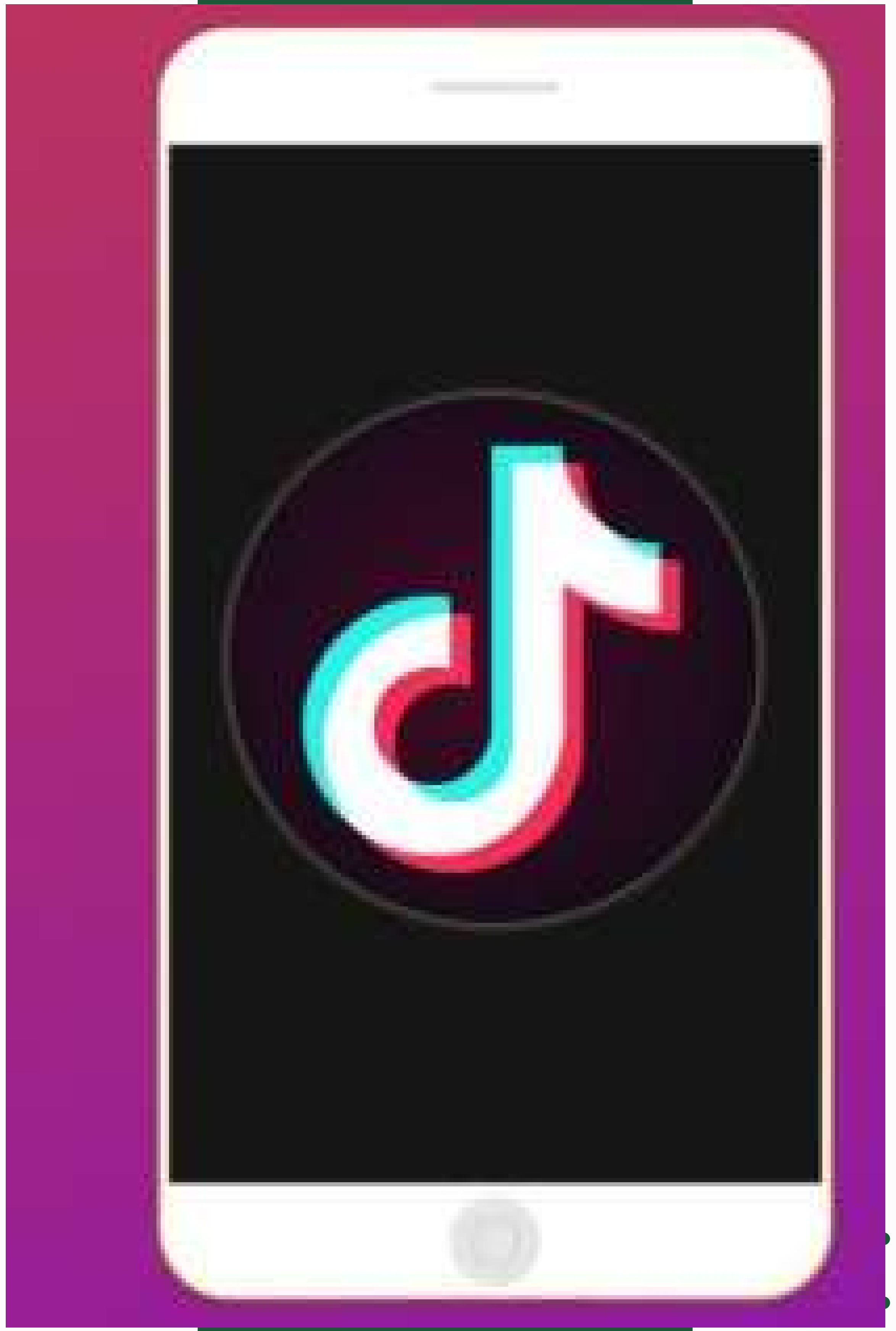
by: Ana Farida

CLASSIFYING VIDEOS



Project Tiktok

- TikTok users can report videos they believe violate the platform's terms of service. With millions of videos created daily, too many are reported for human moderators to review individually.
- As part of its fact-checking initiative will involve the creation of a machine-learning model, which will classify videos into claims or opinions. The model will assign a lower priority to opinion videos for human review and further classify claim videos—for example, by the number of reports—so as to prioritize the most important ones for review.

Classifying
Claim vs. Opinion Videos

Predicting
Verified vs. Unverified Accounts

A/B Test
Comparing Verified vs. Unverified Accounts

A/B Test

Comparing Average View Counts: Verified vs. Unverified Accounts

```
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)
```

```
data.isna().sum()

#                             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
```

```
data1=data.dropna(axis=0).reset_index()
data1.isna().sum()

index                       0
#                           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
```

- Do videos from verified accounts and videos unverified accounts have different average view counts?
- Is there a relationship between the account being verified and the associated videos' view counts?

```
data1.groupby("verified_status")["video_view_count"].mean()

verified_status
not verified    265663.785339
verified         91439.164167
Name: video_view_count, dtype: float64
```

1. Descriptive Statistics:

- Calculate the average (mean) values of video_view_count for each group of verified_status to identify initial differences.
- It appears that videos from not_verified accounts have a higher number of views on average.
- However, this difference could be due to random chance. To confirm, perform a two-sample t-test to check if the difference is statistically significant.

```
verified = data1[data1["verified_status"] == 'verified']["video_view_count"]
not_verified = data1[data1["verified_status"] == 'not verified']["video_view_count"]

stats.ttest_ind(a=not_verified, b=verified, equal_var=False)

Ttest_indResult(statistic=25.499441780633777, pvalue=2.6088823687177823e-120)
```

2. Hypothesis Test:

- **Null hypothesis (H0):** There is no difference in number of views between TikTok videos posted by verified accounts and TikTok videos posted by unverified accounts (any observed difference in the sample data is due to chance or sampling variability).

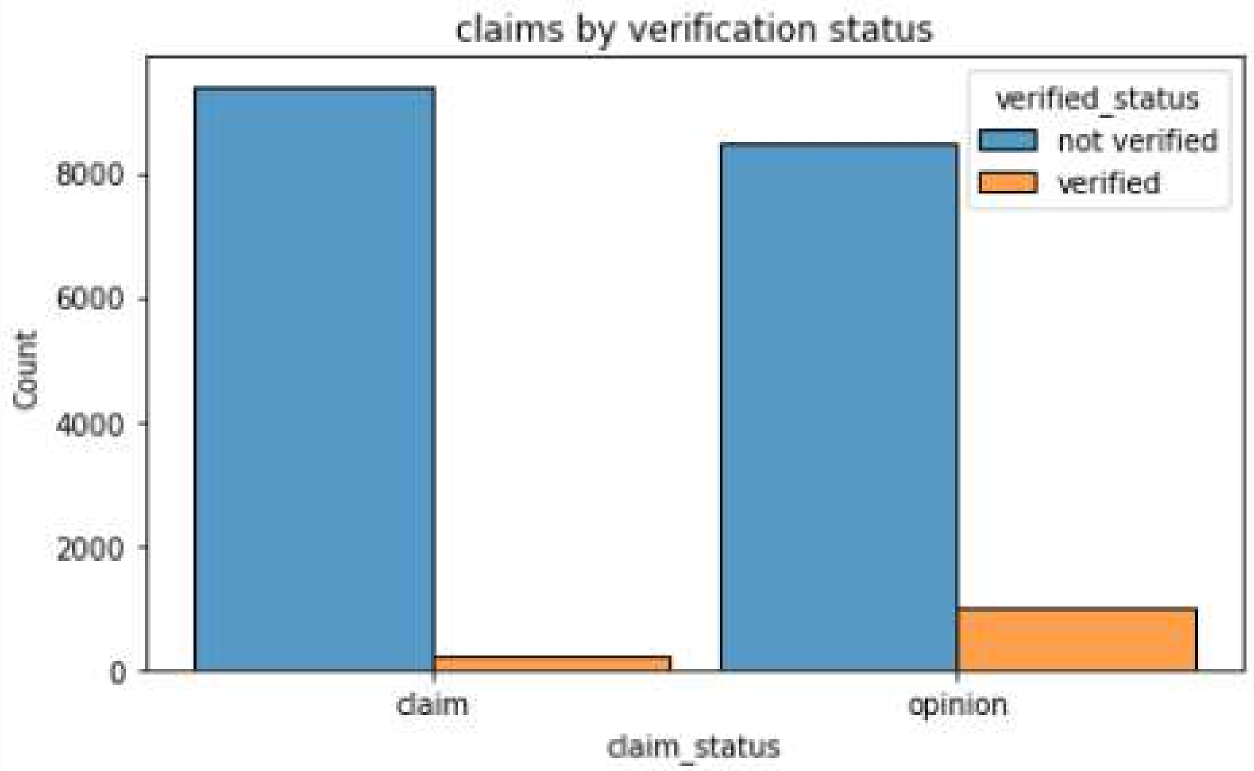
- **Alternative hypothesis (HA):** There is a difference in number of views between TikTok videos posted by verified accounts and TikTok videos posted by unverified accounts (any observed difference in the sample data is due to an actual difference in the corresponding population means).

3. Conduct the t-test:

Since the p-value is extremely small (much smaller than the significance level of 5%), it rejects the null hypothesis. It concludes that **there is a statistically significant difference in the mean video view count between verified and unverified accounts on TikTok.**

Predicting Verified Accounts - 1

- **The purpose of this model is to understand how video features relate to verified users.**

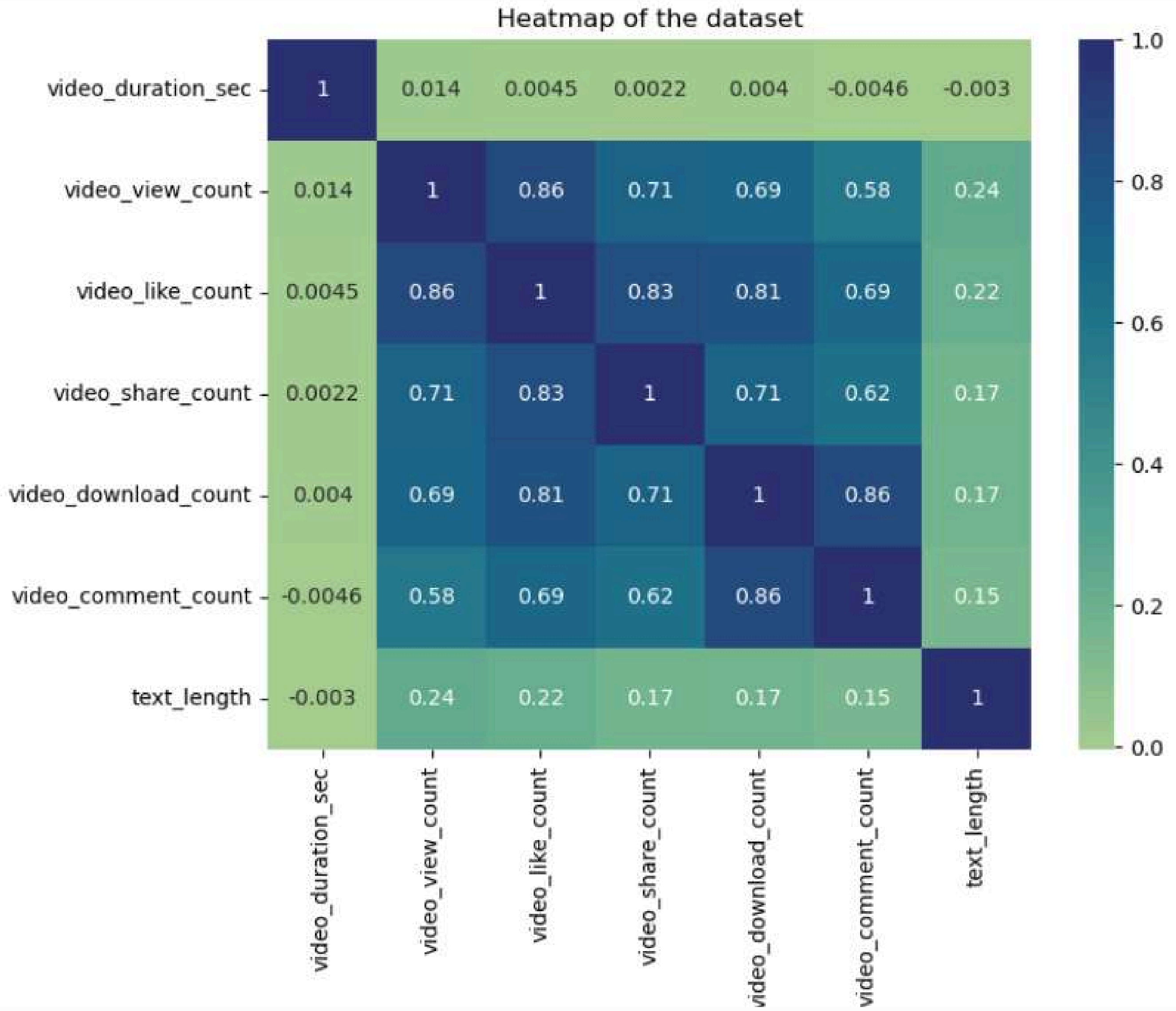


- **Exploratory data analysis shows that verified users are more likely to post opinions.** In order to explore into how video features relate to verified users, perform **a logistic regression with verified status as the target.** The results could help in improving the model for predicting whether a video is a claim or an opinion.

```
data1["verified_status"].value_counts(normalize=True)

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

- Approximately 93.7% of the dataset represents videos posted by unverified accounts and 6.3% represents videos posted by verified accounts. So **the outcome variable is imbalanced.**

- **Correlation matrix** to help determine most correlated variables.



- Logistic regression assumes no strong multicollinearity between features.

- In this data set, **video_view_count and video_like_count are highly correlated (0.86).** In order to meet the assumption, you could **delete video_like_count** and stay with video_view_count, video_share_count, video_download_count, and video_comment_count as features for the video metrics.

Predicting Verified Accounts - 2

- **Use resampling** to create class balance in the outcome variable.

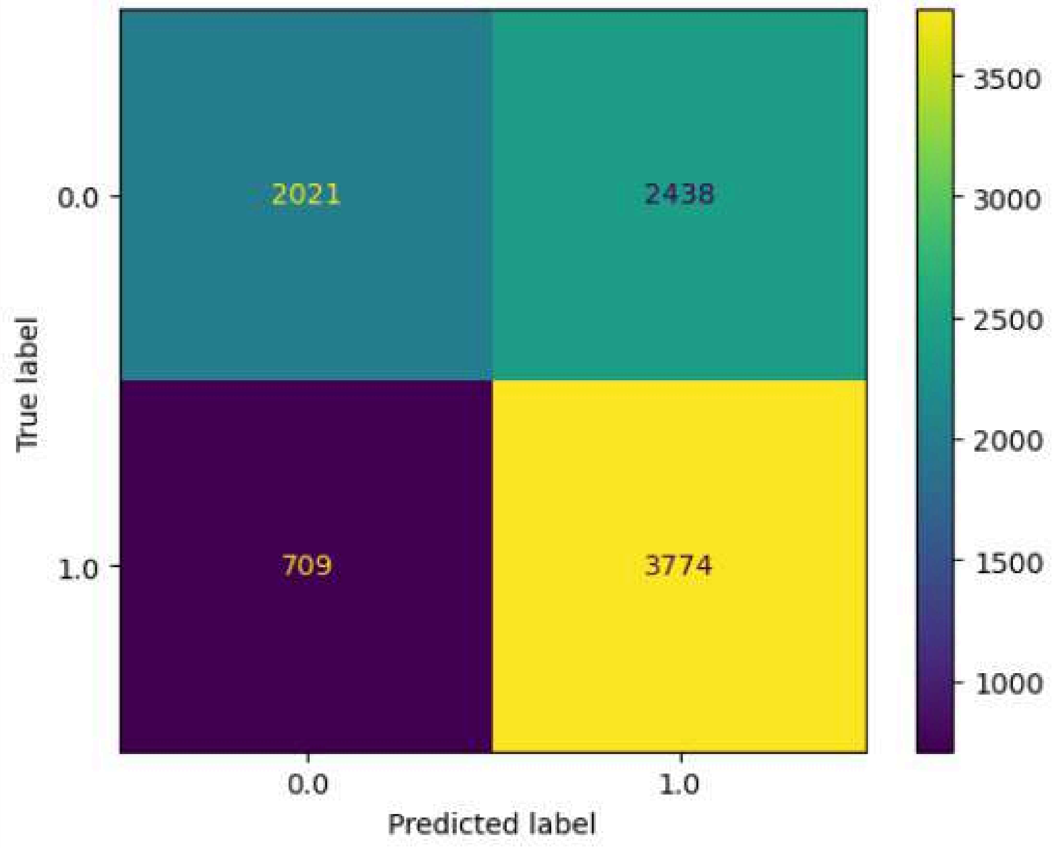
```
majority = data1[data1["verified_status"] == "not verified"]
minority = data1[data1["verified_status"] == "verified"]
minority_upsampled = resample(minority, replace=True, n_samples=len(majority), random_state=0)
upsampled = pd.concat([majority, minority_upsampled]).reset_index(drop=True)
upsampled["verified_status"].value_counts()
```

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

- **Logistic Regression model.**

```
y = upsampled["verified_status"]
X = upsampled[["video_duration_sec", "claim_status", "author_ban_status", "video_view_count", "video_share_count",
               "video_download_count", "video_comment_count"]]
```

```
log_clf = LogisticRegression(random_state=0, max_iter=800).fit(X_train_final, y_train_final)
```




```
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      0.45      0.56      4459
not verified       0.61      0.84      0.71      4483

    accuracy                           0.65      8942
   macro avg       0.67      0.65      0.63      8942
weighted avg       0.67      0.65      0.63      8942
```

- **The logistic regression model was able to achieve a precision of 61%, a recall of 84%, and an accuracy of 65%.** These precision and recall values specifically pertain to the prediction of the "not verified" class, which is our target of interest. The "verified" class has different metrics, and the weighted average combines the results for both classes.

	Feature Name	Model Coefficient
0	video_duration_sec	8.493546e-03
1	video_view_count	-2.277453e-06
2	video_share_count	5.458611e-06
3	video_download_count	-2.143023e-04
4	video_comment_count	3.899371e-04
5	claim_status_opinion	3.772015e-04
6	author_ban_status_banned	-1.675961e-05
7	author_ban_status_under review	-7.084767e-07

- **Each additional second of the video_duration_sec is associated with 0.009 increase in the log-odds of the user having a verified status.**
- Other video features have small estimated coefficients in the model, so their association with verified status seems to be small.

Classifying Claim Videos - 1

- The purpose of this model is to increase response time and system efficiency by automating the initial stages of the claims process.
- **The goal of this model is to predict whether a TikTok video presents a "claim" or presents an "opinion".**

- The claim_status column in the data indicates whether a video is a claim or opinion and will serve as the target variable. This is a binary classification task. There are two types of prediction errors:
- False positives: A video predicted as a claim but is actually an opinion.
- False negatives: A video predicted as an opinion but is actually a claim.
- False negatives are more serious because claims violating terms of service may go unreviewed. Misclassifying an opinion as a claim only results in unnecessary review. In order to reduce false negatives, recall will be the key evaluation metric.

The confusion matrix quadrants:

- Upper-left (True Negatives): Opinions correctly classified as opinions.
- Upper-right (False Positives): Opinions misclassified as claims.
- Lower-left (False Negatives): Claims misclassified as opinions.
- Lower-right (True Positives): Claims correctly classified as claims.
- A perfect model would have only true negatives and true positives, with no false negatives or false positives.

- **Random Forest model**

```
rf = RandomForestClassifier(random_state=0)
cv_params = {'max_depth': [5, 7, None],
             'max_features': [0.3, 0.6],
             'max_samples': [0.7],
             'min_samples_leaf': [1,2],
             'min_samples_split': [2,3],
             'n_estimators': [75,100,200],
             }
scoring = {'accuracy', 'precision', 'recall', 'f1'}
rf_cv = GridSearchCV(rf, cv_params, scoring=scoring, cv=5, refit='recall')

rf_cv.fit(X_train_final, y_train)
```

```
y_pred = rf_cv.best_estimator_.predict(X_val_final)
```

```
rf_cv.best_params_

{'max_depth': None,
 'max_features': 0.6,
 'max_samples': 0.7,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'n_estimators': 200}
```

```
rf_cv.best_score_

0.9948228253467271
```

- **XGBoost model**

```
xgb = XGBClassifier(objective='binary:logistic', random_state=0)
cv_params = {'max_depth': [4, 12],
             'min_child_weight': [3, 5],
             'learning_rate': [0.01, 0.1],
             'n_estimators': [20]
             }
scoring = {'accuracy', 'precision', 'recall', 'f1'}
xgb_cv = GridSearchCV(xgb, cv_params, scoring=scoring, cv=5, refit='recall')
xgb_cv.fit(X_train_final, y_train)
```

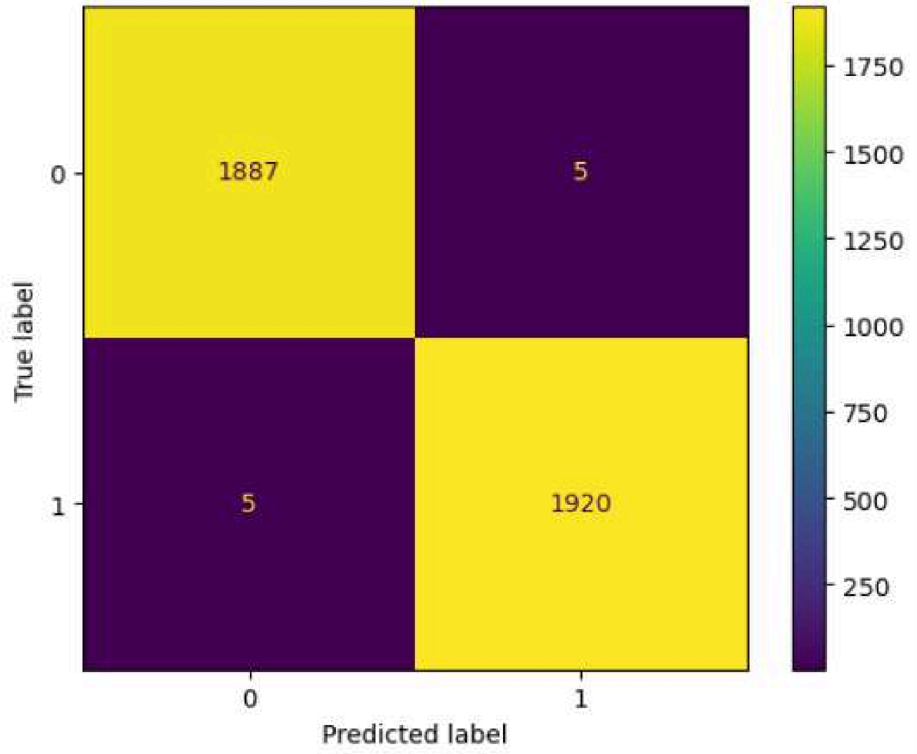
```
y_pred = xgb_cv.best_estimator_.predict(X_val_final)
```

```
xgb_cv.best_params_

{'learning_rate': 0.1,
 'max_depth': 4,
 'min_child_weight': 3,
 'n_estimators': 20}
```

```
xgb_cv.best_score_

0.9899903287480573
```

Classifying Claim Videos - 2

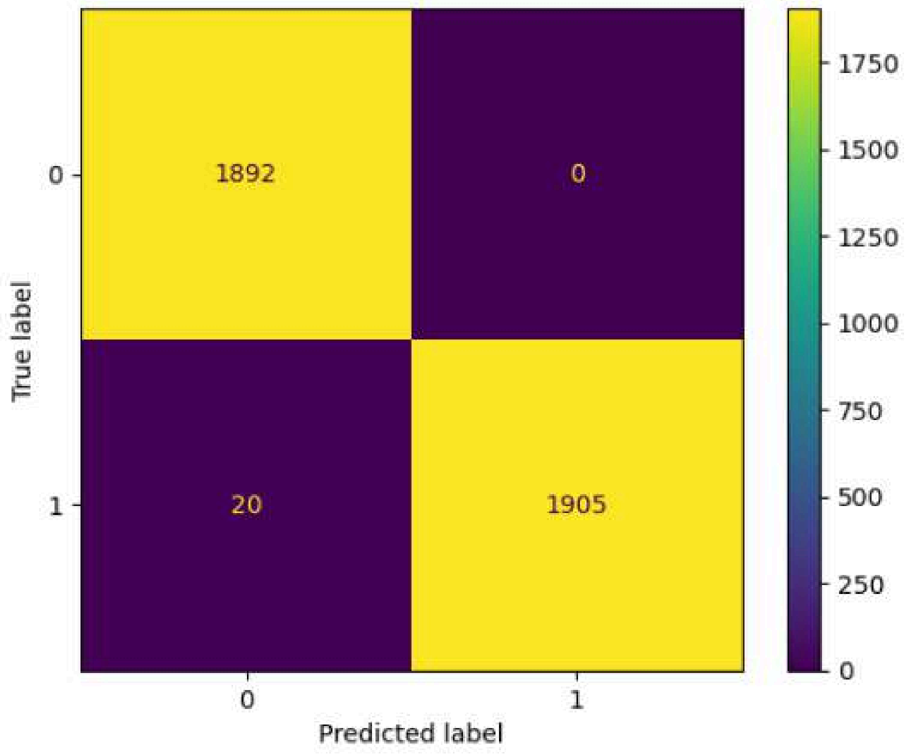
- **Result Random Forest model to predict on validation data.**



- Random Forest model performed almost perfectly, with only 10 misclassifications: 5 false positives and 5 false negatives.

	precision	recall	f1-score	support
opinion	1.00	1.00	1.00	1892
claim	1.00	1.00	1.00	1925
accuracy			1.00	3817
macro avg	1.00	1.00	1.00	3817
weighted avg	1.00	1.00	1.00	3817

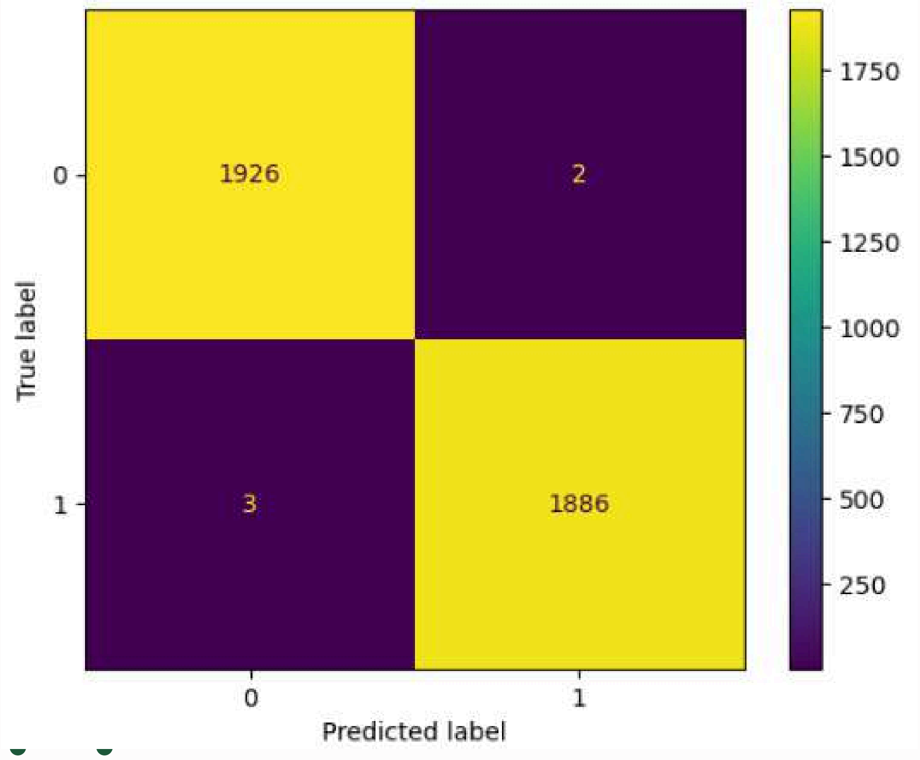
- **Result XGBoost model to predict on validation data.**



- The XGBoost model performed well but made more false negatives. Since identifying claims is the priority, the random forest model, with its better recall, is the better choice.

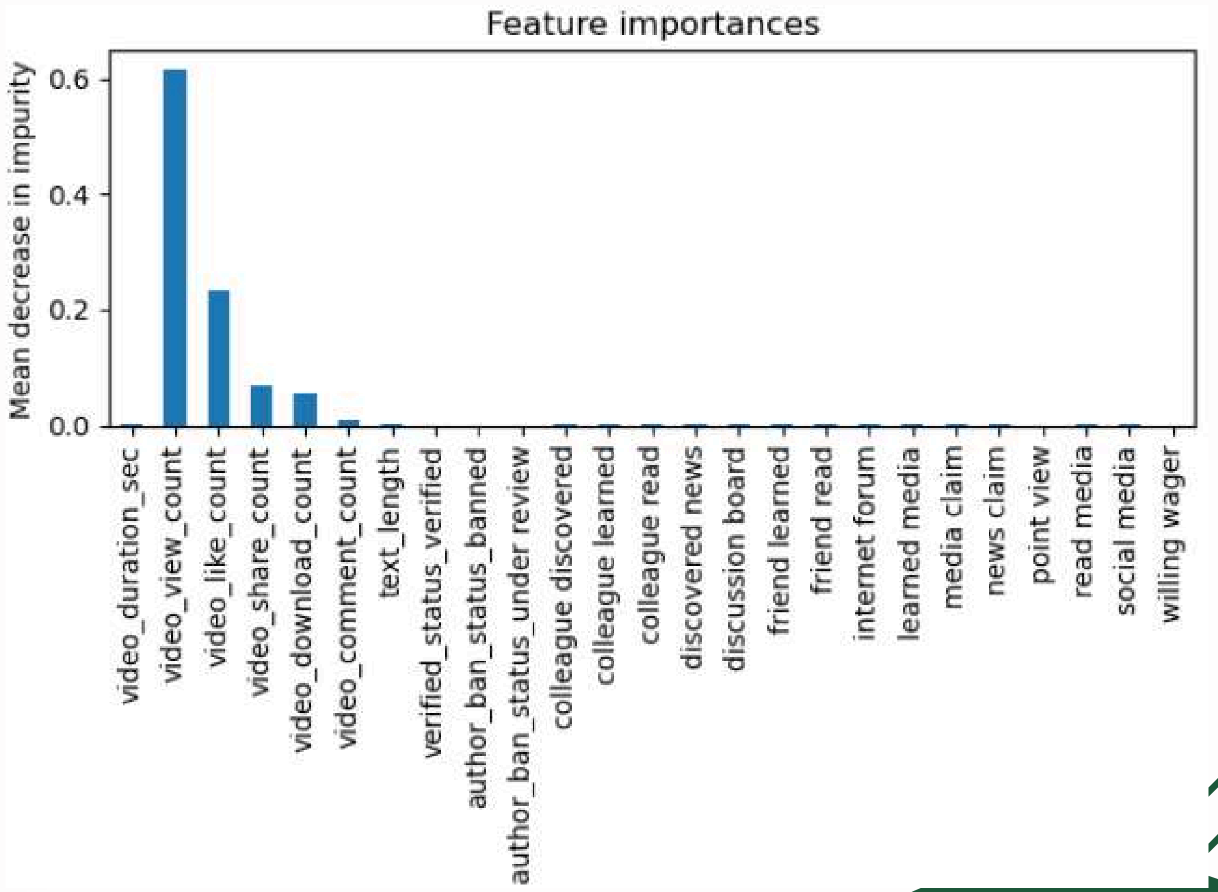
	precision	recall	f1-score	support
opinion	0.99	1.00	0.99	1892
claim	1.00	0.99	0.99	1925
accuracy			0.99	3817
macro avg	0.99	0.99	0.99	3817
weighted avg	0.99	0.99	0.99	3817

- **Use Random Forest model to predict on test data**



- **Random forest model performed well on both validation and test holdout data.** This model does not produce many false negatives. The model classified the claims and opinions very successfully.

- **The model's most predictive features were all related to the user engagement levels** associated with each video. It was classifying videos based on how many **views, likes, shares, and downloads** they received.



DETAILS

Exploratory Data Analysis - 1

Column name	Type	Description
#	int	TikTok assigned number for video with claim/opinion.
claim_status	obj	Whether the published video has been identified as an "opinion" or a "claim." In this dataset, an "opinion" refers to an individual's or group's personal belief or thought. A "claim" refers to information that is either unsourced or from an unverified source.
video_id	int	Random identifying number assigned to video upon publication on TikTok.
video_duration_sec	int	How long the published video is measured in seconds.
video_transcription_text	obj	Transcribed text of the words spoken in the published video.
verified_status	obj	Indicates the status of the TikTok user who published the video in terms of their verification, either "verified" or "not verified."
author_ban_status	obj	Indicates the status of the TikTok user who published the video in terms of their permissions: "active," "under scrutiny," or "banned."
video_view_count	float	The total number of times the published video has been viewed.
video_like_count	float	The total number of times the published video has been liked by other users.
video_share_count	float	The total number of times the published video has been shared by other users.
video_download_count	float	The total number of times the published video has been downloaded by other users.
video_comment_count	float	The total number of comments on the published video.

```
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)
```

```
data.isna().sum()

#                             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
```

- The dataset represents TikTok videos and their metadata. It has 19,382 rows (observations) and 12 columns, comprising a mix of five float64s, three int64s, and four object/text/categorical columns for descriptive information. Some variables have missing values, including claim_status, video_transcription_text, and all of the count variables (views, likes, shares, downloads, and comments).

	#	video_id	video_duration_sec	video_view_count	video_like_count	video_share_count	video_download_count	video_comment_count
count	19382.000000	1.938200e+04	19382.000000	19084.000000	19084.000000	19084.000000	19084.000000	19084.000000
mean	9691.500000	5.627454e+09	32.421732	254708.558688	84304.636030	16735.248323	1049.429627	349.312146
std	5595.245794	2.536440e+09	16.229967	322893.280814	133420.546814	32036.174350	2004.299894	799.638865
min	1.000000	1.234959e+09	5.000000	20.000000	0.000000	0.000000	0.000000	0.000000
25%	4846.250000	3.430417e+09	18.000000	4942.500000	810.750000	115.000000	7.000000	1.000000
50%	9691.500000	5.618664e+09	32.000000	9954.500000	3403.500000	717.000000	46.000000	9.000000
75%	14536.750000	7.843960e+09	47.000000	504327.000000	125020.000000	18222.000000	1156.250000	292.000000
max	19382.000000	9.999873e+09	60.000000	999817.000000	657830.000000	256130.000000	14994.000000	9599.000000

- **Summary statistics**
- Many of the count variables seem to have outliers at the high end of the distribution. They have very large standard deviations and maximum values that are very high compared to their quartile values.

Exploratory Data Analysis - 2


```
data["claim_status"].value_counts()
```

```
claim      9608
opinion    9476
```

- This dataset contains 19,084 claim_status samples, with the number of claim and opinion videos being nearly balanced, There are 9,608 claim video, which is just more than 50% of the total number.

```
claims = data[data["claim_status"] == "claim"]
print(f'Mean view count claims: {claims["video_view_count"].mean()}')
print(f'Median view count claims: {claims["video_view_count"].median()}')
```

```
Mean view count claims: 501029.4527477102
Median view count claims: 501555.0
```

```
opinions = data[data["claim_status"] == "opinion"]
print(f'Mean view count opinion: {opinions["video_view_count"].mean()}')
print(f'Median view count opinion: {opinions["video_view_count"].median()}')
```

```
Mean view count opinion: 4956.43224989447
Median view count opinion: 4953.0
```

- For videos with a "claim" status and a "opinion" status, the average (mean) and the middle value (median) of the view count are similar, indicating the data is fairly balanced without extreme skewness within each category.

claim_status	author_ban_status	#
claim	active	6566
	banned	1439
	under review	1603
opinion	active	8817
	banned	196
	under review	463

- **Combination of claim status and author ban status:**

- There are more banned authors of claim videos than opinion videos, likely because claim videos are more highly regulated. Those posting claims must adhere to a much stricter set of rules than those who post opinions.
- It's, however, unclear whether the claim videos lead to more bans or if the claim authors are more likely to break the rules.
- This data is useful to demonstrate trends for banned versus active authors, but not that a specific video caused a ban. It's also possible that banned authors have written videos that did not violate any of the given rules.

Exploratory Data Analysis - 3

	video_view_count			video_like_count			video_share_count		
	count	mean	median	count	mean	median	count	mean	median
author_ban_status									
active	15383	215927.039524	8616.0	15383	71036.533836	2222.0	15383	14111.466164	437.0
banned	1635	445845.439144	448201.0	1635	153017.236697	105573.0	1635	29998.942508	14468.0
under review	2066	392204.836399	365245.5	2066	128718.050339	71204.5	2066	25774.696999	9444.0

- **The engagement level associated with each different claim status:**

- Banned authors and under review authors get more views, likes, and shares than active authors.
- In most groups, the mean is much greater than the median, which indicates that there are some videos with very high engagement counts.

Create three new columns to help better understand engagement rates:

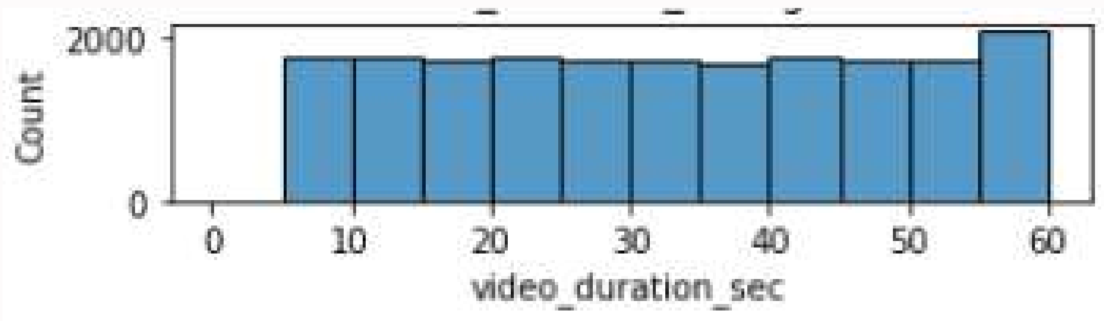
- `likes_per_view`: represents the number of likes divided by the number of views for each video
- `comments_per_view`: represents the number of comments divided by the number of views for each video
- `shares_per_view`: represents the number of shares divided by the number of views for each video

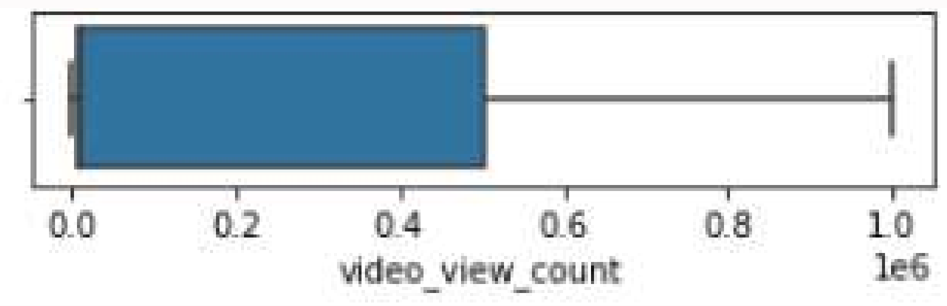
```
data["likes_per_view"] = data["video_like_count"] / data["video_view_count"]
data["comments_per_view"] = data["video_comment_count"] / data["video_view_count"]
data["shares_per_view"] = data["video_share_count"] / data["video_view_count"]
```

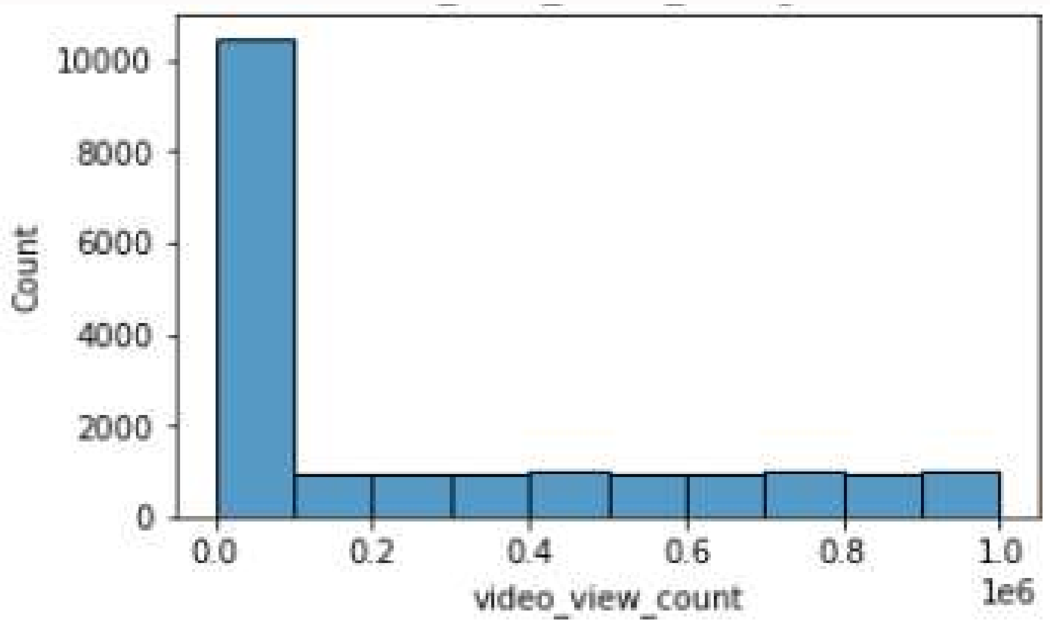
		likes_per_view			comments_per_view			shares_per_view		
		count	mean	median	count	mean	median	count	mean	median
claim_status	author_ban_status									
claim	active	6566	0.329542	0.326538	6566	0.001393	0.000776	6566	0.065456	0.049279
	banned	1439	0.345071	0.358909	1439	0.001377	0.000746	1439	0.067893	0.051606
	under review	1603	0.327997	0.320867	1603	0.001367	0.000789	1603	0.065733	0.049967
opinion	active	8817	0.219744	0.218330	8817	0.000517	0.000252	8817	0.043729	0.032405
	banned	196	0.206868	0.198483	196	0.000434	0.000193	196	0.040531	0.030728
	under review	463	0.226394	0.228051	463	0.000536	0.000293	463	0.044472	0.035027

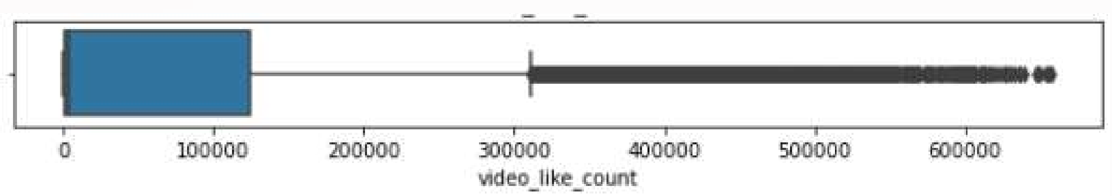
- Videos by banned and under-review authors generally receive more views, likes, and shares than those by non-banned authors. However, the engagement rate—defined as likes, comments, and shares per view—varies more based on whether the video is a claim or opinion rather than the ban status of an author.
- Claim videos tend to get more views, likes, and shares than opinion videos, showing they are generally more engaging and well-liked.
- Banned authors' videos get slightly more views and likes compared with active or reviewed authors when it comes to claim videos; however, when it comes to opinion videos, active, and under_review authors seem to have higher engagement rates from all metrics.
- **There is a substantial correlation between claim status and engagement level.**

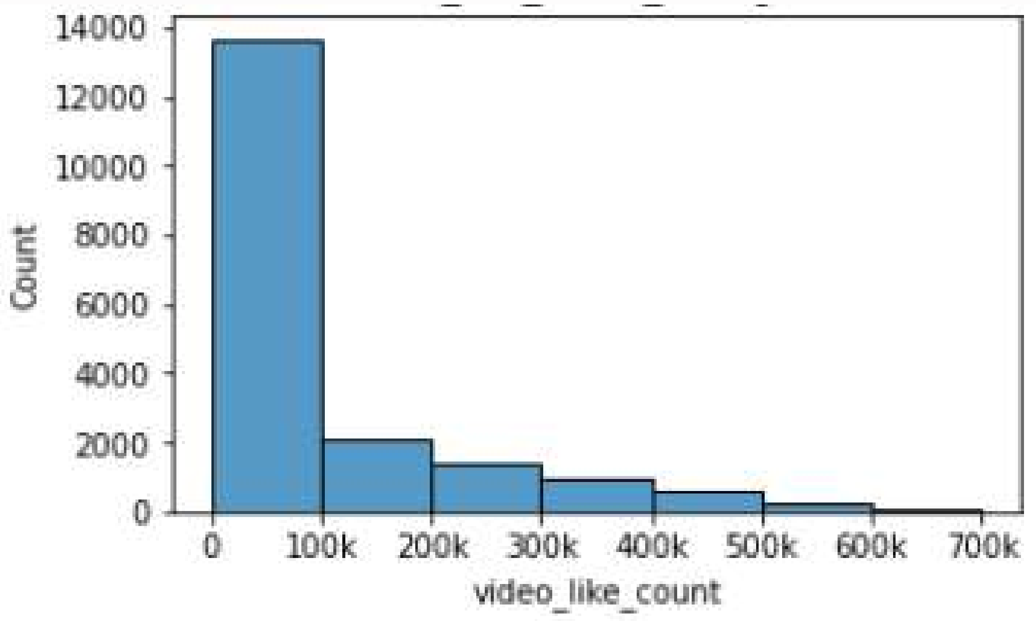
Exploratory Data Analysis - 4











Video_duration:

- **All videos duration are 5-60 seconds**, and the distribution is uniform.

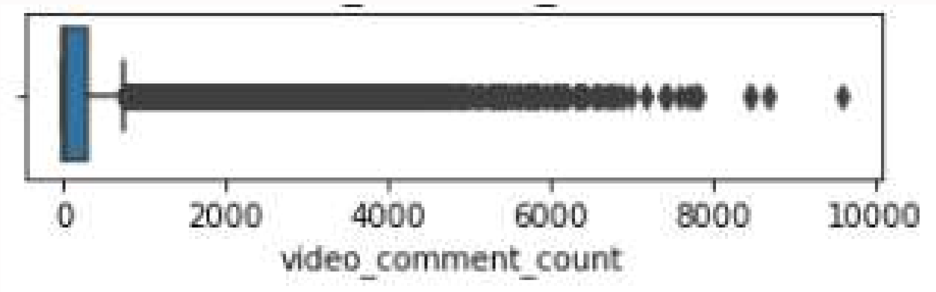
Video_view_count:

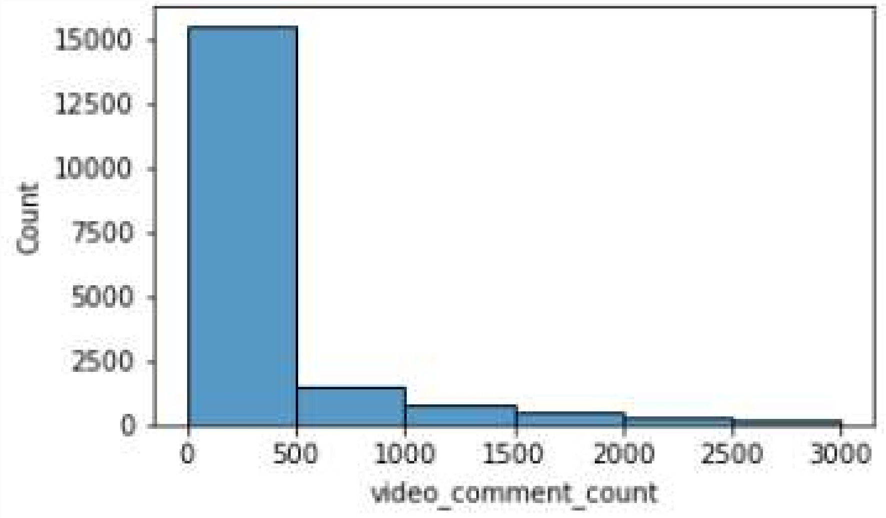
- It has a very imbalance distribution, with **more than half the videos receiving fewer than 100,000 views.** Distribution of view counts > 100,000 views is uniform.

Video_like_count:

- Similar to view count, there are far **more videos with < 100,000 likes** than there are videos with more. The data skews right, with many videos at the upper extremity of like count.

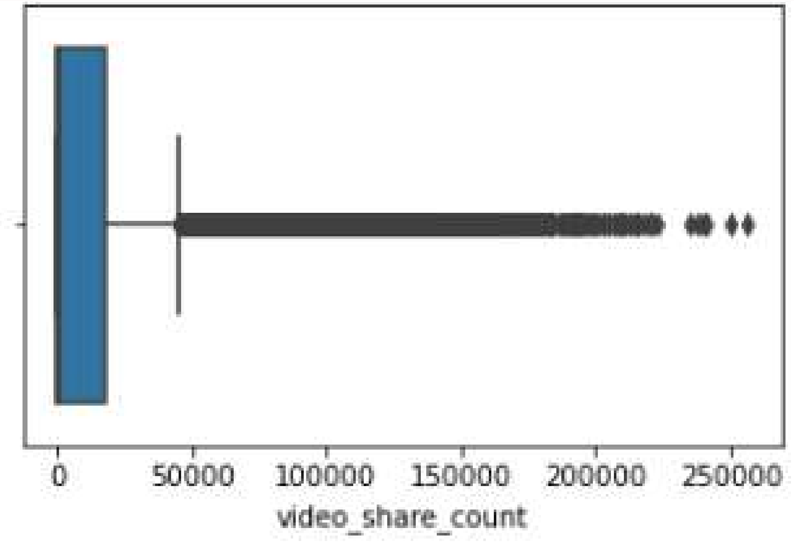
Exploratory Data Analysis - 5

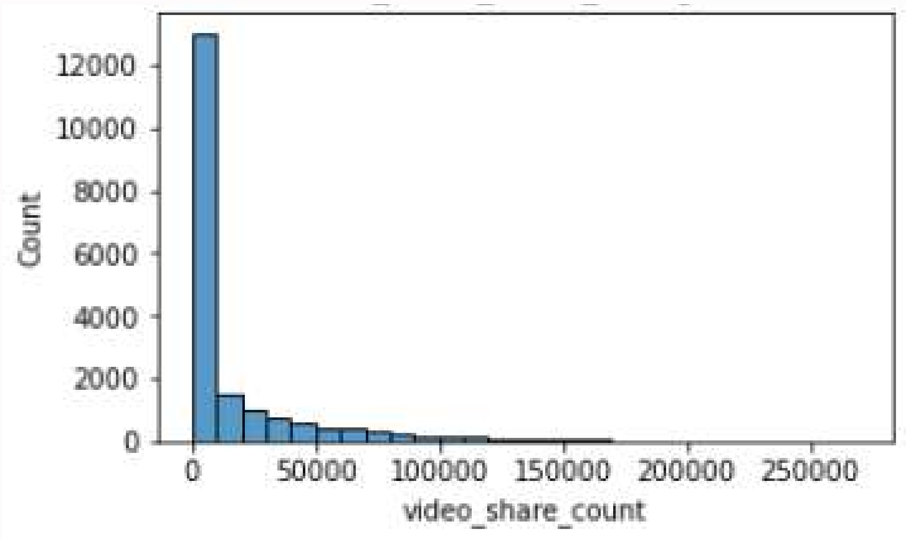




Video_comment_count:

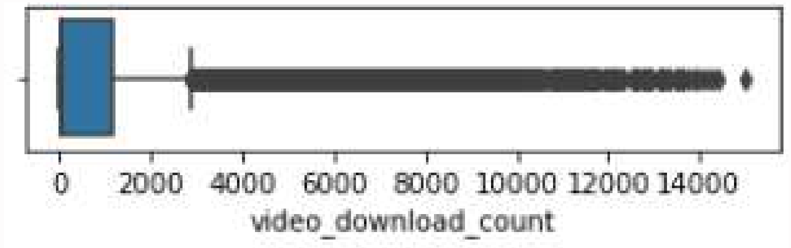
- The majority of videos are grouped at the bottom of the range of values. The distribution is very right-skewed. **The videos have fewer than 100 comments.**

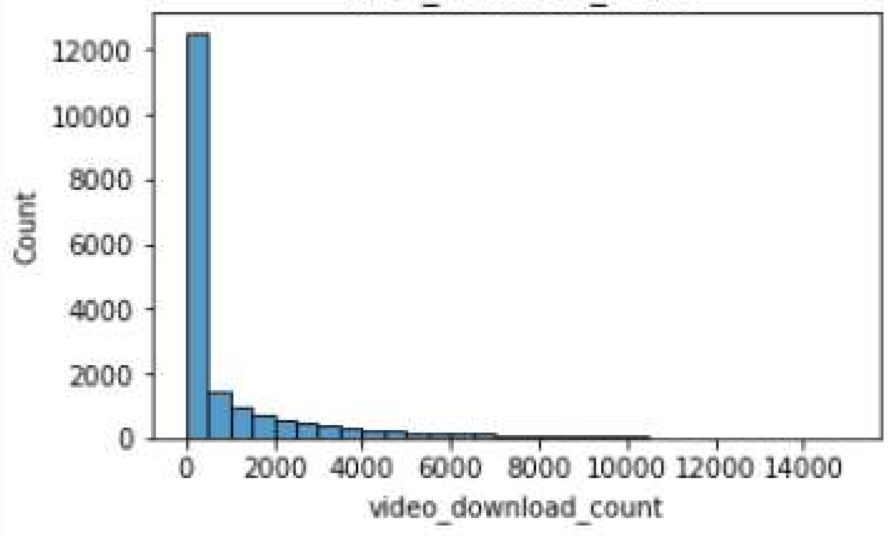




Video_share_count:

- **The majority of videos had fewer than 10,000 shares**. The distribution is very skewed to the right.

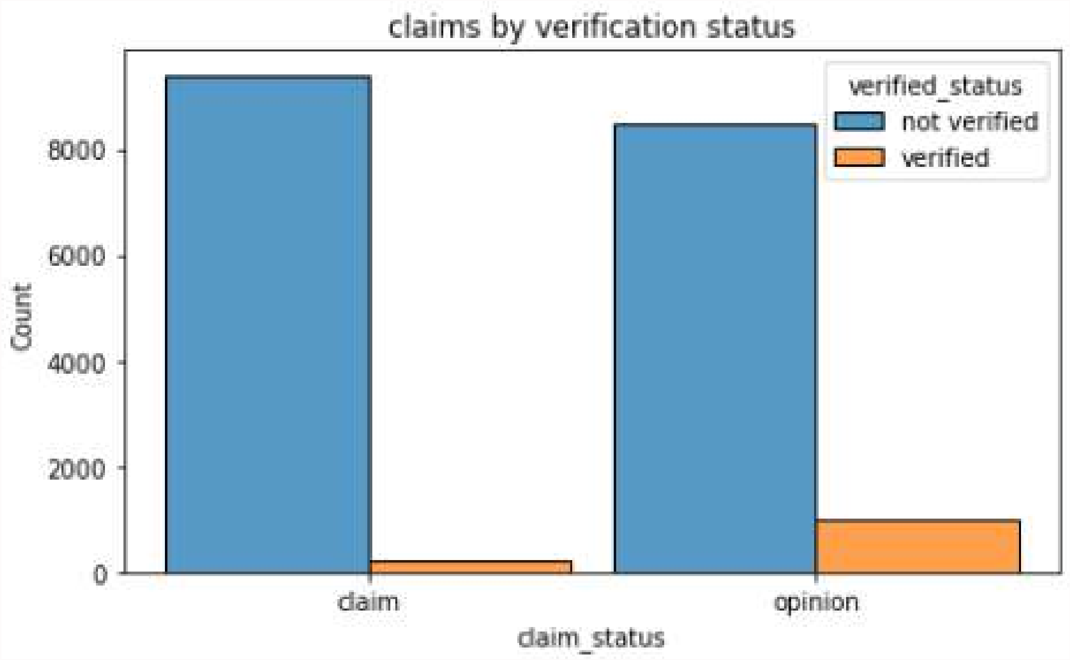




Video_download_count:

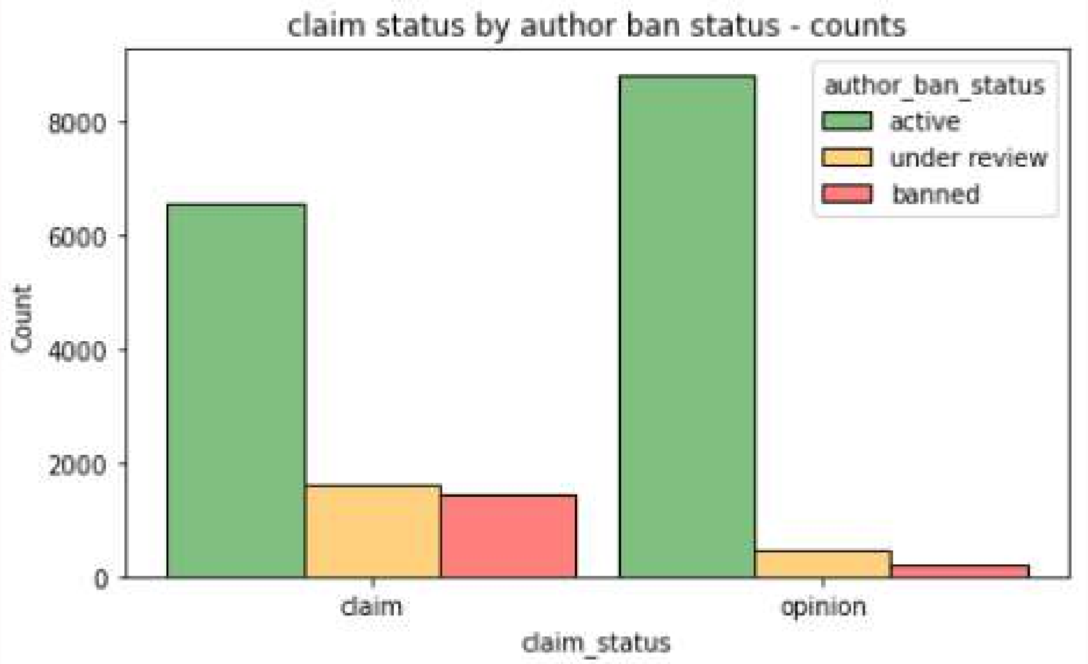
- **The majority of videos were downloaded fewer than 500 times,** but some were downloaded over 12,000 times. The data is very skewed to the right.

Exploratory Data Analysis - 6



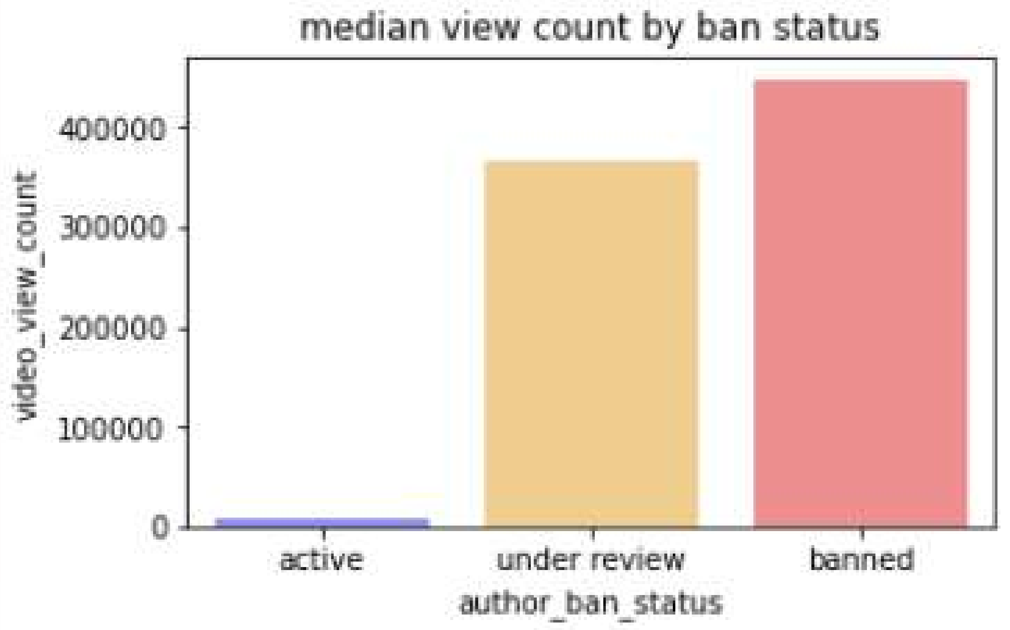
Claims by verification status:

- **There are fewer verified users than unverified users.** If a user is verified, they are much more likely to post opinions.



Claim status by author ban status:

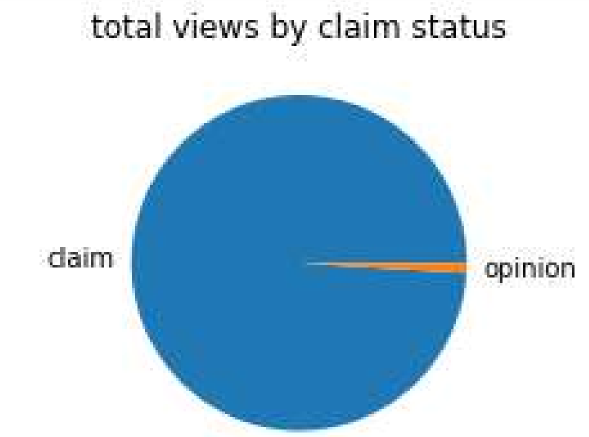
- **For both claims and opinions, there are many more active authors than banned authors or authors under review.** However, the proportion of active authors is far greater for opinion videos than for claim videos. It seems that authors who post claim videos are more likely to come under review and/or get banned.



Median view count by ban status:

- **Median view counts for non-active authors (under review and banned) are many times greater than active authors.** Since non-active authors are more likely to post claims, and that videos by non-active authors get far more views on aggregate than videos by active authors, then video_view_count might be a good indicator of claim status.

```
data.groupby("claim_status")["video_view_count"].median()

claim_status
claim      501555.0
opinion      4953.0
Name: video_view_count, dtype: float64
```



Median and total video_view_counts:

- **It is dominated by claim videos.**

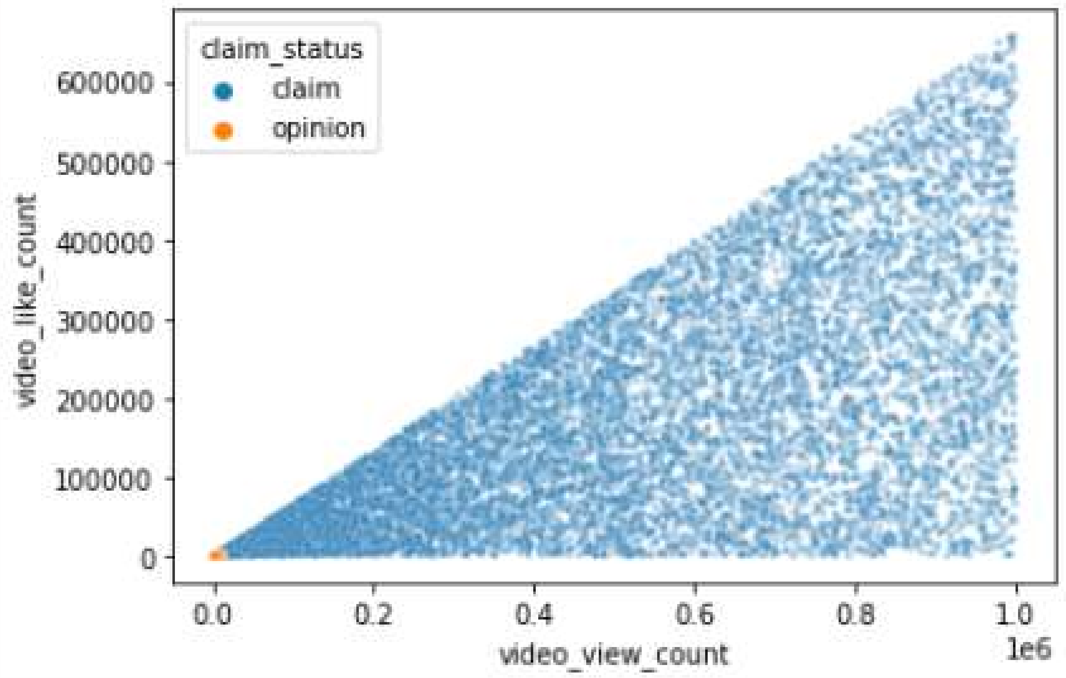
Exploratory Data Analysis - 7

- **Number of outliers for video engagement level (views, likes, shares, downloads, and comments).**
    - Each category has a similar number of outliers, roughly 3,000 in total.

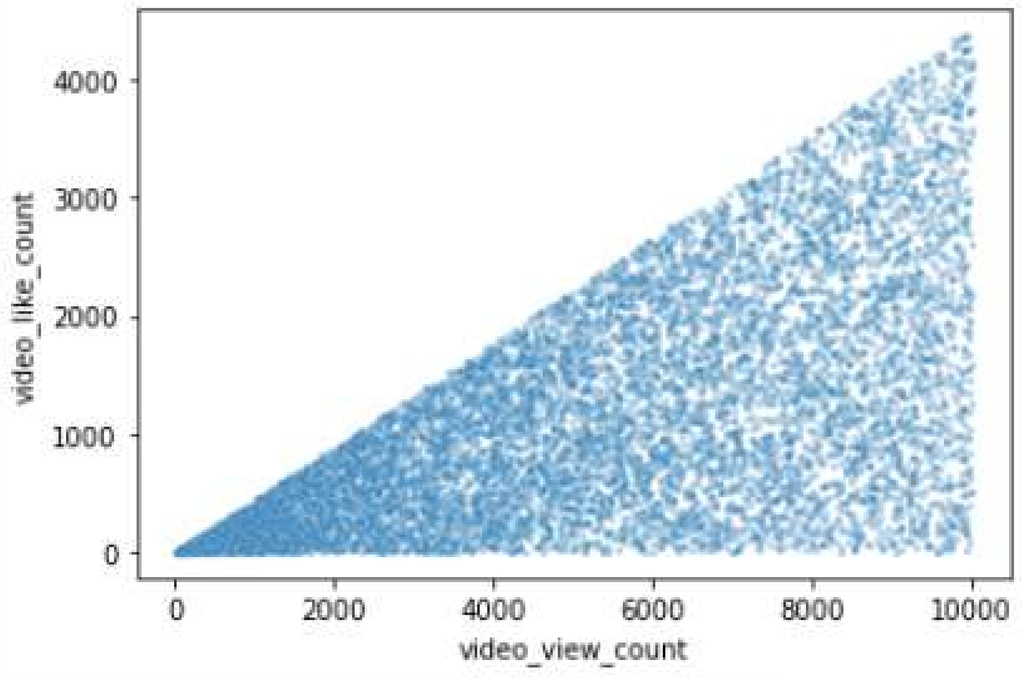
```python
count_cols = ["video_view_count", "video_like_count", "video_share_count", "video_download_count",
              "video_comment_count"]

for col in count_cols:
    q1 = data[col].quantile(0.25)
    q3 = data[col].quantile(0.75)
    iqr = q3 - q1
    median = data[col].median()
    outlier_treshold = median + 1.5*iqr

    outlier_count = (data[col] > outlier_treshold).sum()
    print(f'Number of outliers, {col}:', outlier_count)
```

```
Number of outliers, video_view_count: 2343
Number of outliers, video_like_count: 3468
Number of outliers, video_share_count: 3732
Number of outliers, video_download_count: 3733
Number of outliers, video_comment_count: 3882
```

- **`video_view_count` versus `video_like_count` for claims and opinions.**



- **`video_view_count` versus `video_like_count` for opinions only.**



- **There is a strong relationship between video views and likes.** For both claim and opinion videos, as views increase, likes also tend to increase.

Regression Modeling - 1

A. Exploratory Data Analysis

```
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)
```

```
data.isna().sum()

#                             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
```

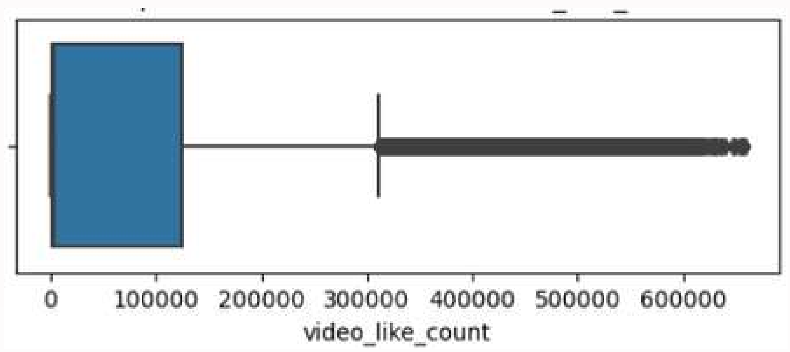
```
data1=data.dropna(axis=0).reset_index()
data1.isna().sum()

index                       0
#                           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
```

- There are very few missing values relative to the number of samples in the dataset. Therefore, **observations with missing values can be dropped.**

```
data.duplicated().sum()

0
```

- **There are no duplicate observations in the data.**

- **Handle outliers for video_like_count**




```
percentile25 = data1["video_like_count"].quantile(0.25)
percentile75 = data1["video_like_count"].quantile(0.75)
iqr = percentile75 - percentile25
upper_limit = percentile75 + (1.5 * iqr)

data1.loc[data1["video_like_count"]>upper_limit, "video_like_count"] = upper_limit
```

```
data1["verified_status"].value_counts(normalize=True)

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

- Approximately 93.7% of the dataset represents videos posted by unverified accounts and 6.3% represents videos posted by verified accounts. So **the outcome variable is imbalanced.**

```
majority = data1[data1["verified_status"] == "not verified"]
minority = data1[data1["verified_status"] == "verified"]
minority_upsampled = resample(minority, replace=True, n_samples=len(majority), random_state=0)
upsampled = pd.concat([majority, minority_upsampled]).reset_index(drop=True)
upsampled["verified_status"].value_counts()

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

- **Use resampling** to create class balance in the outcome variable

Regression Modeling - 2

- **Get the mean `video_transcription_text` length** for videos posted by verified accounts and unverified accounts.

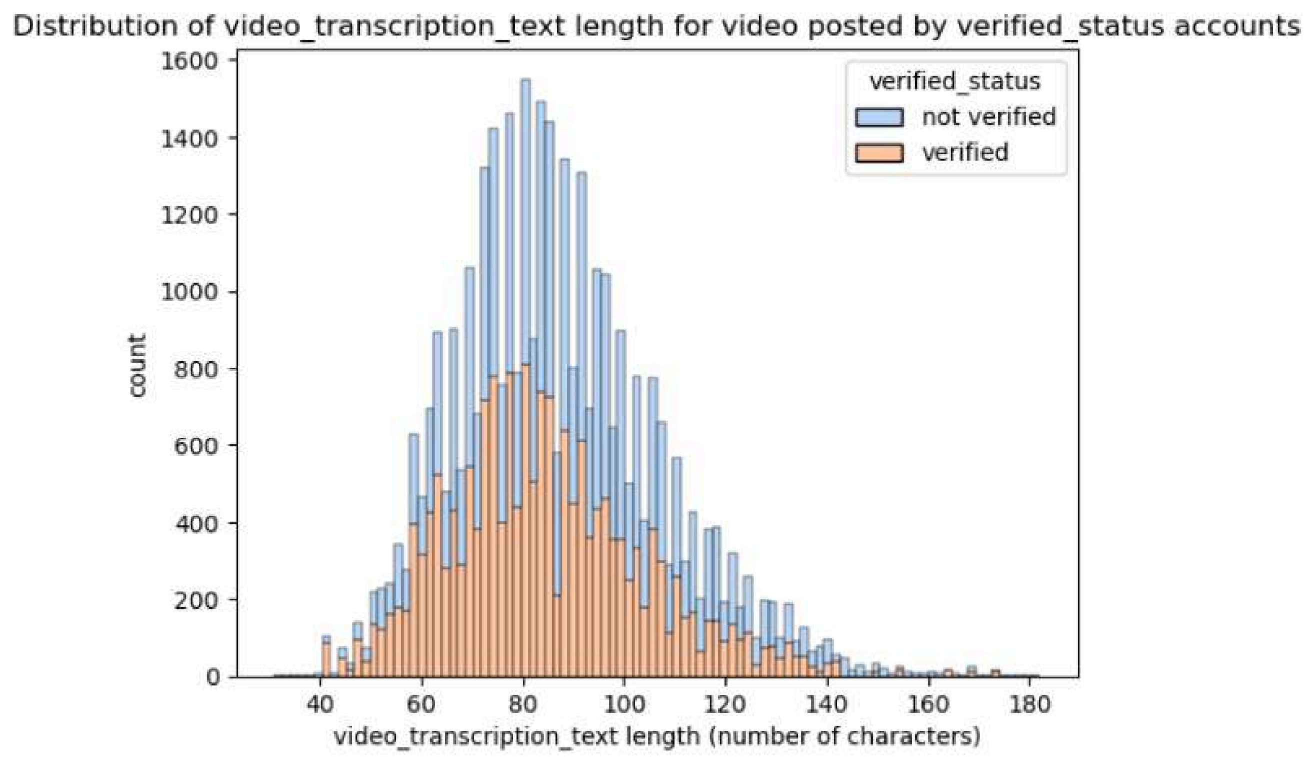
```
upsampled[["verified_status", "video_transcription_text"]].groupby(by="verified_status")[["video_transcription_text"]].agg(func=lambda array: np.mean([len(text) for text in array]))
```

verified_status	video_transcription_text
not verified	89.401141
verified	84.569559

- **Extract the length of each `video_transcription_text`** and add this as a column to the dataframe.

```
upsampled["text_length"] = upsampled["video_transcription_text"].apply(func=lambda text: len(text))
```

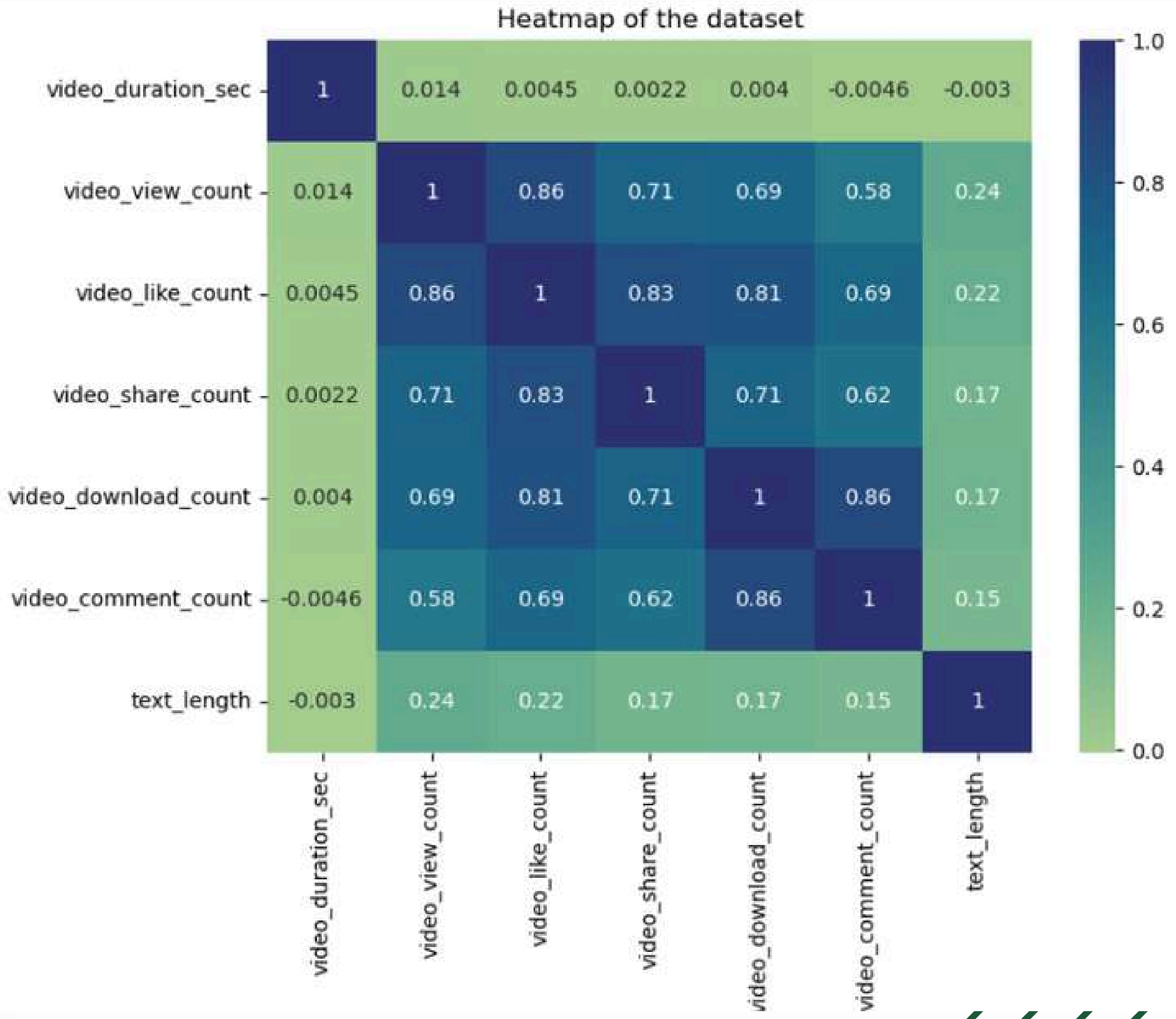
- **Distribution of video_transcription_text length:**



- **Correlation matrix** to help determine most correlated variables.

- Logistic regression assumes no strong multicollinearity between features.

- In this data set, video_view_count and video_like_count are highly correlated (0.86). In order to meet the assumption, you could delete video_like_count and stay with video_view_count, video_share_count, video_download_count, and video_comment_count as features for the video metrics.



Regression Modeling - 3

B. Construct Model

- Set y and X variables.

```
y = upsampled["verified_status"]
X = upsampled[["video_duration_sec", "claim_status", "author_ban_status", "video_view_count", "video_share_count",
                "video_download_count", "video_comment_count"]]
```

- Split the data into training and testing sets.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((26826, 7), (8942, 7), (26826,), (8942,))
```

- There are 7 features align between the training and testing sets.
- The number of rows aligns between the features and the outcome variable for training (`26826`) and testing (`8942`).

- Use one-hot encoding to convert the claim_status and author_ban_status features.

```
X_train_to_encode = X_train[["claim_status", "author_ban_status"]]
X_encoder = OneHotEncoder(drop='first', sparse_output=False)
X_train_encoded = X_encoder.fit_transform(X_train_to_encode)

X_train_encoded_df = pd.DataFrame(data=X_train_encoded, columns=X_encoder.get_feature_names_out())
X_train.drop(columns=["claim_status", "author_ban_status"])
X_train_final = pd.concat([X_train.drop(columns=["claim_status",
                        "author_ban_status"]).reset_index(drop=True), X_train_encoded_df], axis=1)
```

- Encode categorical values of the outcome variable the training set using one-hot encoding.

```
y_encoder = OneHotEncoder(drop='first', sparse_output=False)
y_train_final = y_encoder.fit_transform(y_train.values.reshape(-1, 1)).ravel()
```

- 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)
```

Regression Modeling - 4

C. Result and Evaluation

- **Transform the testing features using the encoder.**

```
X_test_to_encode = X_test[["claim_status", "author_ban_status"]]
X_test_encoded = X_encoder.transform(X_test_to_encode)

X_test_encoded_df = pd.DataFrame(data=X_test_encoded, columns=X_encoder.get_feature_names_out())
X_test.drop(columns=["claim_status", "author_ban_status"])
X_test_final = pd.concat([X_test.drop(columns=["claim_status", "author_ban_status"]).reset_index(drop=True), X_test_encoded_df], axis=1)
```

- **Use the logistic regression model to get predictions on the encoded testing set.**

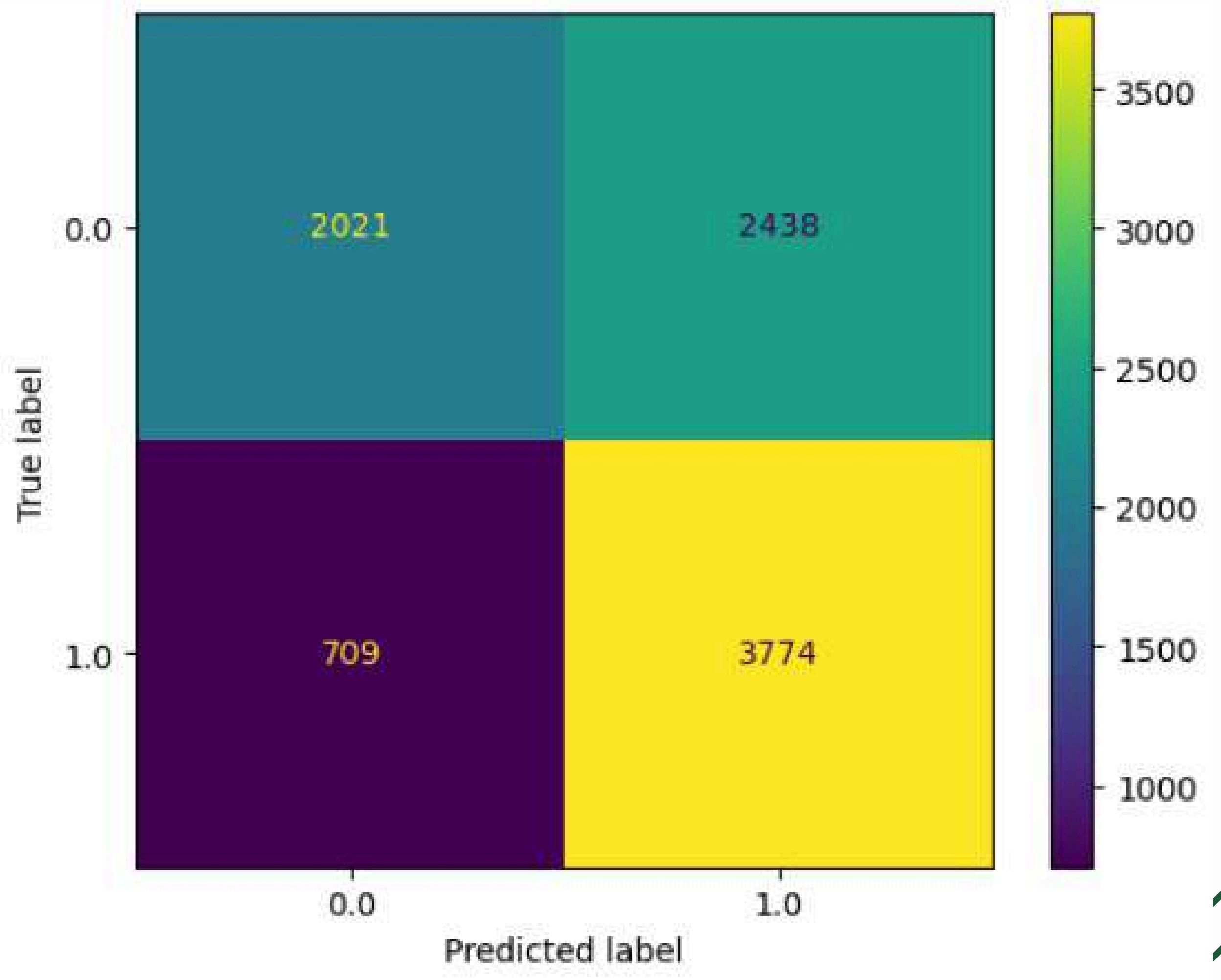
```
y_pred = log_clf.predict(X_test_final)
y_test_final = y_encoder.transform(y_test.values.reshape(-1, 1))
```

- **Get shape of each training and testing set**

```
X_train_final.shape, y_train_final.shape, X_test_final.shape, y_test_final.shape

((26826, 8), (26826,), (8942, 8), (8942, 1))
```

- The number of features (`8`) aligns between the training and testing sets.
- The number of rows aligns between the features and the outcome variable for training (`26826`) and testing (`8942`).

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



Regression Modeling - 5

- **Create a classification report to evaluate the performance of the logistic regression model.**

```
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      0.45      0.56      4459
not verified       0.61      0.84      0.71      4483

    accuracy                           0.65      8942
   macro avg       0.67      0.65      0.63      8942
weighted avg       0.67      0.65      0.63      8942
```

- **The logistic regression model was able to achieve a precision of 61%, a recall of 84%, and an accuracy of 65%.** These precision and recall values specifically pertain to the prediction of the "not verified" class, which is our target of interest. The "verified" class has different metrics, and the weighted average combines the results for both classes.

- **Get the model coefficients (which represent log-odds ratios)**

	Feature Name	Model Coefficient
0	video_duration_sec	8.493546e-03
1	video_view_count	-2.277453e-06
2	video_share_count	5.458611e-06
3	video_download_count	-2.143023e-04
4	video_comment_count	3.899371e-04
5	claim_status_opinion	3.772015e-04
6	author_ban_status_banned	-1.675961e-05
7	author_ban_status_under review	-7.084767e-07

- **Each additional second of the video_duration_sec is associated with 0.009 increase in the log-odds of the user having a verified status.**
- Other video features have small estimated coefficients in the model, so their association with verified status seems to be small.

Tree-Based Modeling - 1

A. Exploratory Data Analysis

```
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)
```

```
data.isna().sum()

#                             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
```

```
data1=data.dropna(axis=0).reset_index()
data1.isna().sum()

index                       0
#                           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
```

- There are very few missing values relative to the number of samples in the dataset. Therefore, **observations with missing values can be dropped.**

```
data.duplicated().sum()

0
```

- **There are no duplicate observations in the data.**
- Tree-based models can handle outliers well, so there's **no need to remove or impute values based on their distribution.**

```
data["claim_status"].value_counts(normalize=True)

claim_status
claim      0.503458
opinion    0.496542
Name: proportion, dtype: float64
```

- Approximately 50.3% of the dataset represents claims and 49.7% represents opinions, so the outcome variable is balanced.

- **Extract the length (character count) and add this to the dataframe as `text_length.**

```
data['text_length'] = data['video_transcription_text'].str.len()
```

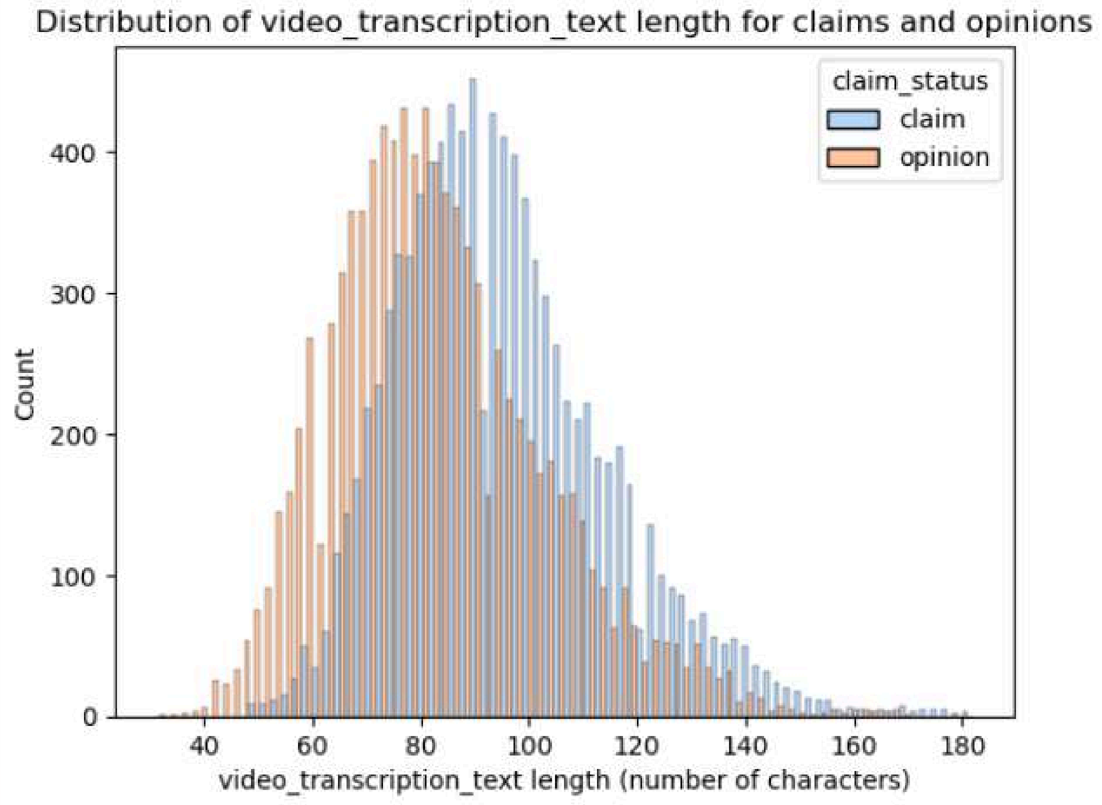
- **Get the mean `text_length`** for claims and opinions.

```
data[['claim_status', 'text_length']].groupby('claim_status').mean()
```

claim_status	text_length
claim	95.376978
opinion	82.722562

- Letter count distributions for both claims and opinions are approximately normal with a slight right skew. Claim videos tend to have more characters about 13 more on average

- **Distribution of text length:**



Tree-Based Modeling - 2

B. Construct Model

- Set y and X variables.

```
X = data.copy()
X = X.drop(['#', 'video_id'], axis=1)
X['claim_status'] = X['claim_status'].replace({'opinion': 0, 'claim': 1})
X = pd.get_dummies(X,
                   columns=['verified_status', 'author_ban_status'],
                   drop_first=True)

X = X.drop(['claim_status'], axis=1)
y = X['claim_status']
```

- Split the data into training and testing sets.

```
X_tr, X_test, y_tr, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
X_train, X_val, y_train, y_val = train_test_split(X_tr, y_tr, test_size=0.25, random_state=0)
X_train.shape, X_val.shape, X_test.shape, y_train.shape, y_val.shape, y_test.shape

((11450, 11), (3817, 11), (3817, 11), (11450,), (3817,), (3817,))
```

- First, split data into training and testing sets, 80/20. Then, divide the training set into training and validation sets, 75/25, to result in a final ratio of 60/20/20 for train/validate/test sets.
- The number of features (`11`) aligns between the training and testing sets.
- The number of rows aligns between the features and the outcome variable for training (`11,450`) and both validation and testing data (`3,817`).

- Convert a collection of text from video_transcription_text column to a matrix of token counts for training data (`X_train_final`).

```
count_vec = CountVectorizer(ngram_range=(2, 3),
                            max_features=15,
                            stop_words='english')
count_data = count_vec.fit_transform(X_train['video_transcription_text']).toarray()
count_df = pd.DataFrame(data=count_data, columns=count_vec.get_feature_names_out())
X_train_final = pd.concat([X_train.drop(columns=['video_transcription_text']).reset_index(drop=True), count_df], axis=1)
```

- Convert a collection of text from video_transcription_text column to a matrix of token counts for tvalidation data (`X_val_final`).

```
validation_count_data = count_vec.transform(X_val['video_transcription_text']).toarray()
validation_count_df = pd.DataFrame(data=validation_count_data, columns=count_vec.get_feature_names_out())
X_val_final = pd.concat([X_val.drop(columns=['video_transcription_text']).reset_index(drop=True), validation_count_df], axis=1)
```

Tree-Based Modeling - 3

B. Construct Model

- **Set y and X variables.**

```
X = data.copy()
X = X.drop(['#', 'video_id'], axis=1)
X['claim_status'] = X['claim_status'].replace({'opinion': 0, 'claim': 1})
X = pd.get_dummies(X,
                   columns=['verified_status', 'author_ban_status'],
                   drop_first=True)

X = X.drop(['claim_status'], axis=1)
y = X['claim_status']
```

- **Split the data into training and testing sets.**

```
X_tr, X_test, y_tr, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
X_train, X_val, y_train, y_val = train_test_split(X_tr, y_tr, test_size=0.25, random_state=0)
X_train.shape, X_val.shape, X_test.shape, y_train.shape, y_val.shape, y_test.shape

((11450, 11), (3817, 11), (3817, 11), (11450,), (3817,), (3817,))
```

- First, split data into training and testing sets, 80/20. Then, divide the training set into training and validation sets, 75/25, to result in a final ratio of 60/20/20 for train/validate/test sets.
- The number of features (`11`) aligns between the training and testing sets.
- The number of rows aligns between the features and the outcome variable for training (`11,450`) and both validation and testing data (`3,817`).

- **Convert a collection of text from video_transcription_text column to a matrix of token counts for training data (`X_train_final`).**

```
count_vec = CountVectorizer(ngram_range=(2, 3),
                            max_features=15,
                            stop_words='english')
count_data = count_vec.fit_transform(X_train['video_transcription_text']).toarray()
count_df = pd.DataFrame(data=count_data, columns=count_vec.get_feature_names_out())
X_train_final = pd.concat([X_train.drop(columns=['video_transcription_text']).reset_index(drop=True), count_df], axis=1)
```

- **Convert a collection of text from video_transcription_text column to a matrix of token counts for validation data (`X_val_final`).**

```
validation_count_data = count_vec.transform(X_val['video_transcription_text']).toarray()
validation_count_df = pd.DataFrame(data=validation_count_data, columns=count_vec.get_feature_names_out())
X_val_final = pd.concat([X_val.drop(columns=['video_transcription_text']).reset_index(drop=True), validation_count_df], axis=1)
```

Tree-Based Modeling - 4

- Convert a collection of text from video_transcription_text column to a matrix of token counts for test data (`X_test_final`).

```
test_count_data = count_vec.transform(X_test['video_transcription_text']).toarray()
test_count_df = pd.DataFrame(data=test_count_data, columns=count_vec.get_feature_names_out())
X_test_final = pd.concat([X_test.drop(columns=['video_transcription_text']
                          ).reset_index(drop=True), test_count_df], axis=1)
```

- Evaluate Random Forest model

```
rf = RandomForestClassifier(random_state=0)
cv_params = {'max_depth': [5, 7, None],
             'max_features': [0.3, 0.6],
             'max_samples': [0.7],
             'min_samples_leaf': [1,2],
             'min_samples_split': [2,3],
             'n_estimators': [75,100,200],
             }
scoring = {'accuracy', 'precision', 'recall', 'f1'}
rf_cv = GridSearchCV(rf, cv_params, scoring=scoring, cv=5, refit='recall')

rf_cv.fit(X_train_final, y_train)
```

```
y_pred = rf_cv.best_estimator_.predict(X_val_final)
```

```
rf_cv.best_params_

{'max_depth': None,
 'max_features': 0.6,
 'max_samples': 0.7,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'n_estimators': 200}
```

```
rf_cv.best_score_

0.9948228253467271
```

- Evaluate XGBoost model

```
xgb = XGBClassifier(objective='binary:logistic', random_state=0)
cv_params = {'max_depth': [4, 12],
             'min_child_weight': [3, 5],
             'learning_rate': [0.01, 0.1],
             'n_estimators': [20]
             }
scoring = {'accuracy', 'precision', 'recall', 'f1'}
xgb_cv = GridSearchCV(xgb, cv_params, scoring=scoring, cv=5, refit='recall')
xgb_cv.fit(X_train_final, y_train)
```

```
y_pred = xgb_cv.best_estimator_.predict(X_val_final)
```

```
xgb_cv.best_params_

{'learning_rate': 0.1,
 'max_depth': 4,
 'min_child_weight': 3,
 'n_estimators': 20}
```

```
xgb_cv.best_score_

0.9899903287480573
```

THANK YOU

by: Ana Farida