TikTok Project

In this project I am acting as a data professional at TikTok. My supervisor has expressed the claim that more data and less expertise(that Duck) is crucial for learning models that can used to determine whether a video contains a claim or whether it offers an opinion. With a business proposition like this, how can we ensure the backing of user reports and practical data is not offended.

Classifying videos using machine learning

In this project, I will be using machine learning techniques to predict on a binary outcome variable.

The goal is to be able 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 project has three parts:

Part I: Ethical considerations

- In this course, I prepare for the model to generate false positives rather than false negatives, identifying claims that violate terms of service is crucial, even if it means some opinion videos are mistakenly labeled as claims. The ethical consequences of misclassifying an opinion as a claim is that the video packages human review. However, misclassifying a claim as an opinion allows the claim to remain on the platform, potentially violating TikTok's terms of service and facilitating the dissemination of misinformation.

Part II: Exploratory data analysis

- Perform features selection, extraction, and transformation to prepare the data for modeling.

Part III: Modeling

- Train Decision

Tree models to predict the training target to distinguish between claims and opinions in the data. Evaluate the model on the validation data, which contains zero-shots. Key metrics to keep track of are recall, precision, accuracy, and F1 score.

- Modeling Approach

Using the video variant dataset from the data dictionary as the target variable, the model aims to classify each video as either a claim or an opinion. This constitutes a binary classification task. Both the models, random forest, and adaboost will select

Throughout this project notebook, you'll see references to the problem-solving framework (PACE) framework. The following notebook components are labeled with the respective PACE stage: Plan, Analyze, Construct, and Execute.

PACE: Plan

Reflecting on Evaluation Metrics

To gauge the model's performance effectively, we need to address potential errors. There are two types:

False positives: When the model wrongly identifies a video as a claim instead of an opinion. False negatives: When the model mistakenly labels a video as an opinion instead of a claim.

Considering the ethical implications, minimizing false negatives is crucial. Thus, the chosen evaluation metric is recall.

Developing workflows and model selection process

- Create a function that allows me to generate 5 folds of 20,000 videos in the sample. This is sufficient to conduct a rigorous model validation workflow, broken into the following steps:
- Fit the model and tune hyperparameters on the training set
- Perform feature engineering on the training set and validation set
- Assess the champion model's performance on the test set

Task 1: Imports and data loading

I need to be importing packages needed to conduct the exploratory data modeling tasks to achieve the goal of this project.

```
# Install scikitplot for data manipulation
!pip install scikitplot
import scikitplot as skplt

# Import packages for data manipulation
import numpy as np, pandas as pd, matplotlib.pyplot as plt
import seaborn as sns

# Import packages for data preprocessing
from sklearn.feature_extraction.text import CountVectorizer

# Import packages for data modeling
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import recall_score, precision_score, accuracy_score, f1_score,
                            confusion_matrix, ConfusionMatrixDisplay, precision_score, r
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.tree import plot_tree
from sklearn.metrics import PrecisionRecallDisplay
from sklearn.metrics import RocCurveDisplay
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer

# Import packages for saving models
import joblib
import pickle

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

PACE: Analyze

Task 2: Examine data, summary info, and descriptive stats

Inspect the first five rows of the dataset.

```
# Display first few rows
data.head(5)
```

	claim_status	video_id	video_duration_sec	video_transcription_text	verified_status	author_ban_status	video_view_count	video_like_count	video_share_count	video_download_cou
0	claim	5768674423	53	siamese dressed while she was the kitten	not verified	under review	2343799.0	195279.0	67065.0	7059.0
1	claim	4942865567	33	siamese dressed while she was the kitten	not verified	active	6487702.0	1048616.0	178443.0	58624.0
2	claim	8702456739	51	siamese dressed while she was the kitten	not verified	active	54881750.0	7786623.0	1811077.0	591358.0
3	claim	8968232963	20	siamese dressed while she was the kitten	not verified	active	4807095.0	816004.0	158612.0	51835.0
4	claim	7884657567	43	siamese dressed while she was the kitten	not verified	active	4773843.0	1004463.0	244213.0	12243.0
5	claim	7339561315	35	siamese dressed while she was the kitten	not verified	active	8064516.0	1548616.0	493083.0	81053.0

Get the number of rows and columns in the dataset.

```
# Get number of rows and columns
data.shape
```

```
(19382, 12)
```

Get the data types of the columns.

```
# Get data types of columns
data.dtypes
```

```
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 statistics for the dataset.

```
# Get basic statistics
data.describe
```

	video_id	video_duration_sec	video_view_count	video_like_count	video_share_count	video_download_count	video_comment_count
count	19382.000000	19382.000000	19084.000000	19382.000000	19382.000000	19382.000000	19382.000000
mean	5627454916.988034	32.421732	254708.558688	84304.636030	16735.248323	1049.429627	349.312646
std	5843189709.142960	16.229967	322893.280814	133420.546814	32703.884030	2004.299894	916.695860
min	1234959053.000000	5.000000	20.000000	0.000000	0.000000	0.000000	0.000000
25%	5393085835.000000	18.000000	42138.250000	8445.000000	626.000000	56.000000	7.000000
50%	5837428565.000000	32.000000	220067.500000	57035.000000	4545.500000	171.000000	26.000000
75%	7342674393.250000	47.000000	475488.750000	111974.750000	17051.000000	781.250000	202.000000
max	9999873375.000000	60.000000	999817.000000	657830.000000	256130.000000	14994.000000	9599.000000

Check for and handle missing values.

```
# Check for missing values
data.isna().sum()
```

```
claim_status                  0
video_id                      0
video_duration_sec            0
video_transcription_text      0
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
```

```
# Drop rows with missing values
data.dropna(inplace=True)
```

```
# Display updated info after dropping rows
data.info()
```

```
<class pandas.core.frame.DataFrame>
Int64Index: 19084 entries, 0 to 19381
Data columns (total 12 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   claim_status              19084 non-null  object
 1   video_id                  19084 non-null  int64
 2   video_duration_sec        19084 non-null  int64
 3   video_transcription_text  19084 non-null  object
 4   verified_status           19084 non-null  object
 5   author_ban_status         19084 non-null  object
 6   video_view_count          19084 non-null  float64
 7   video_like_count          19084 non-null  float64
 8   video_share_count         19084 non-null  float64
 9   video_download_count      19084 non-null  float64
 10  video_comment_count       19084 non-null  float64
dtype: int64
```

Check for and handle duplicates.

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

```
0
```

Check class balance.

```
# Check class balance
data["claim_status"].value_counts.normalize.Mean
```

```
claim    0.505608
opinion  0.494392
Name: claim_status
```

Approximately 50% of the dataset represents claims and 49.7% represents opinions, so this outcome variable is balanced.

PACE: Construct

Task 3: Feature engineering

```
# Create a copy of the dataset prior to feature engineering
data2_copy, data2 = data.copy,data.copy(deep=True)
data2.head(5)
```

	claim_status	video_id	video_duration_sec	video_transcription_text	verified_status	author_ban_status	video_view_count	video_like_count	video_share_count	video_download_cou
0	claim	5768674423	53	siamese dressed while she was the kitten	not verified	under review	2343799.0	195279.0	67065.0	7059.0
1	claim	4942865567	33	siamese dressed while she was the kitten	not verified	active	6487702.0	1048616.0	178443.0	58624.0
2	claim	8702456739	51	siamese dressed while she was the kitten	not verified	active	54881750.0	7786623.0	1811077.0	591358.0
3	claim	8968232963	20	siamese dressed while she was the kitten	not verified	active	4807095.0	816004.0	158612.0	51835.0
4	claim	7884657567	43	siamese dressed while she was the kitten	not verified	active	4773843.0	1004463.0	244213.0	12243.0
5	claim	7339561315	35	siamese dressed while she was the kitten	not verified	active	8064516.0	1548616.0	493083.0	81053.0

Calculate the average text_length for claims and opinions.

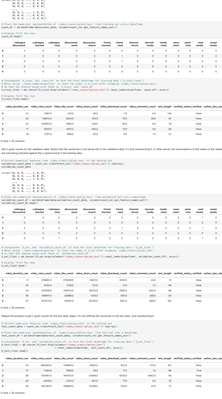
```
# Calculate the average text_length for claims and opinions
data2[["claim_status", "text_length"] ].groupby.mean().round(2)
```

claim_status	text_length
claim	95.376578
opinion	85.488475

Visualize the distribution of `text_length` for claims and opinions.

```
# Visualize the distribution of `text_length` for claims and opinions
# Create a histogram, overlaying both classes
sns.histplot(data=data2, x="text_length", hue="claim_status", bins=30)
plt.title("Distribution of text_length for claims and opinions")
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/matplotlib/backends/backend_agg.py:214:
RuntimeWarning: Glyph 128175 missing from current font.
  font.set_text(s, 0.0, flags=flags)
/usr/local/lib/python3.10/dist-packages/matplotlib/backends/backend_agg.py:183:
RuntimeWarning: Glyph 128175 missing from current font.
  font.get_text(s, 0.0, flags=flags)
```



Feature selection and transformation

Encode target and categorical variables.

```
# Create a copy
data2_copy, data2 = data2.copy,data2.copy(deep=True)
# 0 if "claim", 1 if "opinion"
data2["claim_status"] = np.where(data2["claim_status"] == "claim", 0)
# Dummies for verified_status
data2 = pd.get_dummies(data2, columns=["verified_status", "author_ban_status"],
                       drop_first=True)
data2.head(5)
```

	claim_status	video_id	video_duration_sec	video_transcription_text	video_view_count	video_like_count	video_share_count	video_download_count	video_comment_count	text_length	verified_status_not_	...
0	0	5768674423	53	siamese dressed while she was the kitten	2343799.0	195279.0	67065.0	7059.0	1404.0	100	False	False
1	0	4942865567	33	siamese dressed while she was the kitten	6487702.0	1048616.0	178443.0	58624.0	12444.0	100	False	False
2	0	8702456739	51	siamese dressed while she was the kitten	54881750.0	7786623.0	1811077.0	591358.0	125828.0	100	False	False
3	0	8968232963	20	siamese dressed while she was the kitten	4807095.0	816004.0	158612.0	51835.0	10332.0	100	False	False
4	0	7884657567	43	siamese dressed while she was the kitten	4773843.0	1004463.0	244213.0	12243.0	1828.0	100	False	False
5	0	7339561315	35	siamese dressed while she was the kitten	8064516.0	1548616.0	493083.0	81053.0	12236.0	100	False	False

Task 4: Split the data

In this case, the target variable is `claim_status`.

- Assign the target variable.
- Isolate the features.

```
# Isolate target variable
y = data2["claim_status"]
```

Isolate the features.

```
# Isolate features
X = data2[["video_length", "text_length"]]

# Display first few rows of features dataframe
X.head(5)
```

	video_duration_sec	video_transcription_text	video_view_count	video_like_count	video_share_count	video_download_count	video_comment_count	text_length	verified_status_not_	...
0	53	siamese dressed while she was the kitten	2343799.0	195279.0	67065.0	7059.0	1404.0	100	False	False
1	33	siamese dressed while she was the kitten	6487702.0	1048616.0	178443.0	58624.0	12444.0	100	False	False
2	51	siamese dressed while she was the kitten	54881750.0	7786623.0	1811077.0	591358.0	125828.0	100	False	False
3	20	siamese dressed while she was the kitten	4807095.0	816004.0	158612.0	51835.0	10332.0	100	False	False
4	43	siamese dressed while she was the kitten	4773843.0	1004463.0	244213.0	12243.0	1828.0	100	False	False
5	35	siamese dressed while she was the kitten	8064516.0	1548616.0	493083.0	81053.0	12236.0	100	False	False

Task 5: Tokenize text column

NOTE: You are about to tokenize the text. As you might find it quick and helpful referencing an unfamiliar dataset, bear in mind that the dataset contains 19,382 reviews. Because of the nature of the data provided in this TikTok project, it is best to preserve the video_transcription_text column this makes using lemmatization prohibitively RAM. So, I made the RAM efficient choice of using `CountVectorizer` instead of `TfidfVectorizer`. `CountVectorizer` is a bag of words model, which does not have a focus on word order. This allows for a more manageable number of features than it is possible with `TfidfVectorizer`.

`CountVectorizer` works by splitting text into a grammar, which are groups of consecutive words. For instance, "I drink tea" is a trigram. Setting `ngram_range=(3,3)` means that `CountVectorizer` will only consider trigrams.

```
# Extract the dependent and independent variables
X = data2["video_transcription_text"]
y = data2["claim_status"]

# Set aside 20% of the data for test_holdout and validation_holdout
X_tr, X_test_holdout, y_tr, y_test_holdout = train_test_split(X, y, test_size=0.20,
                                               random_state=42, stratify=y)

# Set aside 20% of the remaining data for validation
X_train, X_val_holdout, y_train, y_val_holdout = train_test_split(X_tr, y_tr, test_size=0.20,
                                               random_state=42, stratify=y_tr)

# Instantiate the vectorizer, ngram_range=(3,3)
vectorizer = CountVectorizer(ngram_range=(3,3), max_features=5000)

# Fit and transform the training, validation, and holdout datasets.
X_train = vectorizer.fit_transform(X_train)
X_val_holdout = vectorizer.transform(X_val_holdout)
X_test_holdout = vectorizer.transform(X_test_holdout)

# Convert to DataFrames.
X_train = pd.DataFrame(X_train.toarray(), columns=vectorizer.get_feature_names_out())
X_val_holdout = pd.DataFrame(X_val_holdout.toarray(), columns=vectorizer.get_feature_names_out)
X_test_holdout = pd.DataFrame(X_test_holdout.toarray(), columns=vectorizer.get_feature_names_out)

# Display the shape of the training set.
X_train.shape
```

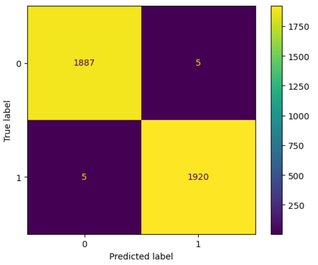
```
(12211, 5000)
```

NOTE: The dataset now has 5,000 features. This is a very high-dimensional dataset. Because the vast majority of these features are zeros, most of the values in the matrix will be sparse (mostly zero). To save memory, we can store the data in a sparse format, which only stores the non-zero values. This will significantly reduce memory usage and speed up processing.

```
# Convert to sparse format
X_train = X_train.sparse.to_dense()
X_val_holdout = X_val_holdout.sparse.to_dense()
X_test_holdout = X_test_holdout.sparse.to_dense()

# Display the first few rows of the training set
X_train.head(5)
```

| | aaaa

```
# Display plot
plt.show()
```



Confusion Matrix Observations:

The upper-left quadrant displays the number of true negatives: the number of opinions that the model accurately classified as so.

The upper-right quadrant displays the number of false positives: the number of opinions that the model misclassified as claims.

The lower-left quadrant displays the number of false negatives: the number of claims that the model misclassified as opinions.

The lower-right quadrant displays the number of true positives: the number of claims that the model accurately classified as so.

A perfect model would yield all true negatives and true positives, and no false negatives or false positives.

As the above confusion matrix shows, this model does not produce any false negatives.

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

In [48]:
```
# Create a classification report
# Create classification report for random forest model
target_labels = ['opinion', 'claim']
print(classification_report(y_val, y_pred, target_names=target_labels))
```

```
              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
```

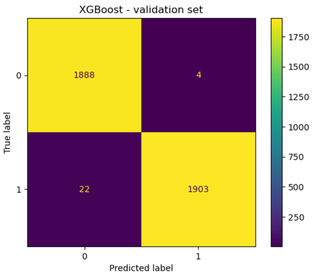
Classification report observations:

The classification report above shows that the random forest model scores were nearly perfect. The confusion matrix indicates that there were 10 misclassifications—five false postives and five false negatives.

XGBoost

In [49]:
```
#Evaluate XGBoost model
y_pred = xgb_cv.best_estimator_.predict(X_val_final)
```

In [50]:
```
# Compute values for confusion matrix
log_cm = confusion_matrix(y_val, y_pred)

# Create display of confusion matrix
log_disp = ConfusionMatrixDisplay(confusion_matrix=log_cm, display_labels=None)

# Plot confusion matrix
log_disp.plot()

# Display plot
plt.title('XGBoost - validation set');
plt.show()
```



In [51]:
```
# Create a classification report
target_labels = ['opinion', 'claim']
print(classification_report(y_val, y_pred, target_names=target_labels))
```

```
              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
```

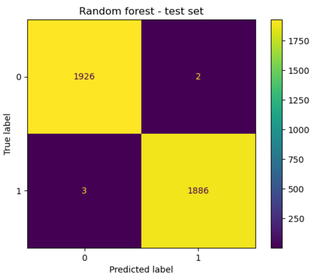
Classification report observations:

The results of the XGBoost model were also nearly perfect. However, its errors tended to be false negatives. Identifying claims was the priority, so it's important that the model be good at capturing all actual claim videos. The random forest model has a better recall score, and is therefore the champion model.

Use champion model to predict on test data

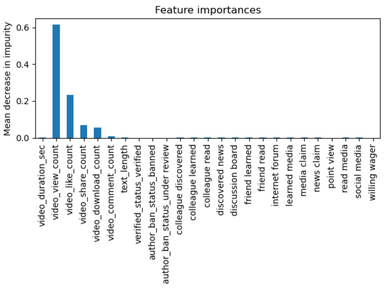
In [52]:
```
# Use champion model to predict on test data
y_pred = rf_cv.best_estimator_.predict(X_test_final)
```

In [53]:
```
# Compute values for confusion matrix
log_cm = confusion_matrix(y_test, y_pred)

# Create display of confusion matrix
log_disp = ConfusionMatrixDisplay(confusion_matrix=log_cm, display_labels=None)

# Plot confusion matrix
log_disp.plot()

# Display plot
plt.title('Random forest - test set');
plt.show()
```



Feature importances of champion model

In [54]:
```
importances = rf_cv.best_estimator_.feature_importances_
rf_importances = pd.Series(importances, index=X_test_final.columns)

fig, ax = plt.subplots()
rf_importances.plot.bar(ax=ax)
ax.set_title('Feature importances')
ax.set_ylabel('Mean decrease in impurity')
fig.tight_layout()
```



Important features observations:

The most predictive features all were related to engagement levels generated by the video. This is not unexpected, as analysis from prior EDA pointed to this conclusion.

Task 8. Conclusion

1. Would you recommend using this model? Why or why not?

Yes, the model is recommended as it demonstrated strong performance on both validation and test data, with consistently high precision and F1 scores. It effectively classified claims and opinions.

2. What was your model doing? Can you explain how it was making predictions?

The model primarily utilized features related to user engagement levels (views, likes, shares, and downloads) associated with each video to make predictions.

3. Are there new features that you can engineer that might improve model performance?

Given the current high performance of the model, there's no immediate need for new feature engineering.

4. What features would you want to have that would likely improve the performance of your model?

While the current model doesn't require additional features, including variables such as the number of times a video was reported and the total user reports for each author could potentially enhance its performance.