Activity_Course 6 TikTok project lab

September 11, 2023

1 TikTok Project Part 4

2 Classifying videos using machine learning

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".

Part 1: Feature engineering

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

Part 2: Modeling

- Build the models, evaluate them, and advise on next steps

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

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

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

# Import packages for data modeling
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import classification_report, accuracy_score,␣
 ↪precision_score, \
recall_score, f1_score, confusion_matrix, ConfusionMatrixDisplay

from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from xgboost import plot_importance
```

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

2.0.1 Examine data, summary info, and descriptive stats

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

[3]:

	#	claim_status	video_id	video_duration_sec	video_transcription_text	verified_status	author_ban_status	video_view_count	video_like_count	video_share_count	video_download_count	video_comment_count
0	1	claim	7017666017	59	someone shared with me that drone deliveries a…	not verified	under review	343296.0	19425.0	241.0	1.0	0.0
1	2	claim	4014381136	32	someone shared with me that there are more mic…	not verified	active	140877.0	77355.0	19034.0	1161.0	684.0
2	3	claim	9859838091	31	someone shared with me that american industria…	not verified	active	902185.0	97690.0	2858.0	833.0	329.0
3	4	claim	1866847991	25	someone shared with me that the metro of st. p…	not verified	active	437506.0	239954.0	34812.0	1234.0	584.0
4	5	claim	7105231098	19	someone shared with me that the number of busi…	not verified	active	56167.0	34987.0	4110.0	547.0	152.0

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

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

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

[5]:
```
#                             int64
claim_status                 object
video_id                      int64
```

```
video_duration_sec            int64
video_transcription_text     object
verified_status              object
author_ban_status            object
video_view_count            float64
video_like_count            float64
video_share_count           float64
video_download_count        float64
video_comment_count         float64
dtype: object
```

[6]:
```
# Get basic information
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19382 entries, 0 to 19381
Data columns (total 12 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   #                         19382 non-null  int64
 1   claim_status              19084 non-null  object
 2   video_id                  19382 non-null  int64
 3   video_duration_sec        19382 non-null  int64
 4   video_transcription_text  19084 non-null  object
 5   verified_status           19382 non-null  object
 6   author_ban_status         19382 non-null  object
 7   video_view_count          19084 non-null  float64
 8   video_like_count          19084 non-null  float64
 9   video_share_count         19084 non-null  float64
 10  video_download_count      19084 non-null  float64
 11  video_comment_count       19084 non-null  float64
dtypes: float64(5), int64(3), object(4)
memory usage: 1.8+ MB
```

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

[7]:

	#	video_id	video_duration_sec	video_view_count	\
count	19382.000000	1.938200e+04	19382.000000	19084.000000	
mean	9691.500000	5.627454e+09	32.421732	254708.558688	
std	5595.245794	2.536440e+09	16.229967	322893.280814	
min	1.000000	1.234959e+09	5.000000	20.000000	
25%	4846.250000	3.430417e+09	18.000000	4942.500000	
50%	9691.500000	5.618664e+09	32.000000	9954.500000	
75%	14536.750000	7.843960e+09	47.000000	504327.000000	
max	19382.000000	9.999873e+09	60.000000	999817.000000	

```
       video_like_count  video_share_count  video_download_count  \
count      19084.000000       19084.000000          19084.000000
mean       84304.636030       16735.248323           1049.429627
std       133420.546814       32036.174350           2004.299894
min            0.000000           0.000000              0.000000
25%          810.750000         115.000000              7.000000
50%         3403.500000         717.000000             46.000000
75%       125020.000000       18222.000000           1156.250000
max       657830.000000      256130.000000          14994.000000

       video_comment_count
count         19084.000000
mean            349.312146
std             799.638865
min               0.000000
25%               1.000000
50%               9.000000
75%             292.000000
max            9599.000000
```

[8]:
```
# Check for missing values
df.isna().sum()
```

[8]:
```
#                           0
claim_status                298
video_id                      0
video_duration_sec            0
video_transcription_text    298
verified_status               0
author_ban_status             0
video_view_count            298
video_like_count            298
video_share_count           298
video_download_count        298
video_comment_count         298
dtype: int64
```

[9]:
```
# Drop rows with missing values
df = df.dropna()
df.isna().sum()
```

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

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

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

[10]:

	#	claim_status	video_id	video_duration_sec	video_transcription_text	verified_status	author_ban_status	video_view_count	video_like_count	video_share_count	video_download_count	video_comment_count
0	1	claim	7017666017	59	someone shared with me that drone deliveries a…	not verified	under review	343296.0	19425.0	241.0	1.0	0.0
1	2	claim	4014381136	32	someone shared with me that there are more mic…	not verified	active	140877.0	77355.0	19034.0	1161.0	684.0
2	3	claim	9859838091	31	someone shared with me that american industria…	not verified	active	902185.0	97690.0	2858.0	833.0	329.0
3	4	claim	1866847991	25	someone shared with me that the metro of st. p…	not verified	active	437506.0	239954.0	34812.0	1234.0	584.0
4	5	claim	7105231098	19	someone shared with me that the number of busi…	not verified	active	56167.0	34987.0	4110.0	547.0	152.0

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

[11]:
```
0
```

[12]:
```
# Check for and handle outliers
# There are many known outliers, but the type of machine learning that will be␣
 ↪used is not significantly impacted by outliers.
```

[13]:
```
# Check class balance
df["claim_status"].value_counts(normalize=True)
```

[13]:
```
claim_status
claim      0.503458
opinion    0.496542
Name: proportion, dtype: float64
```

2.0.2 Feature engineering

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.

[14]:
```
# Extract the length of each `video_transcription_text` and add this as a␣
 ↪column to the dataframe
df['text_length'] = df['video_transcription_text'].str.len()
```

[15]:
```
# Display first few rows of dataframe after adding new column
df.head(5)
```

[15]:

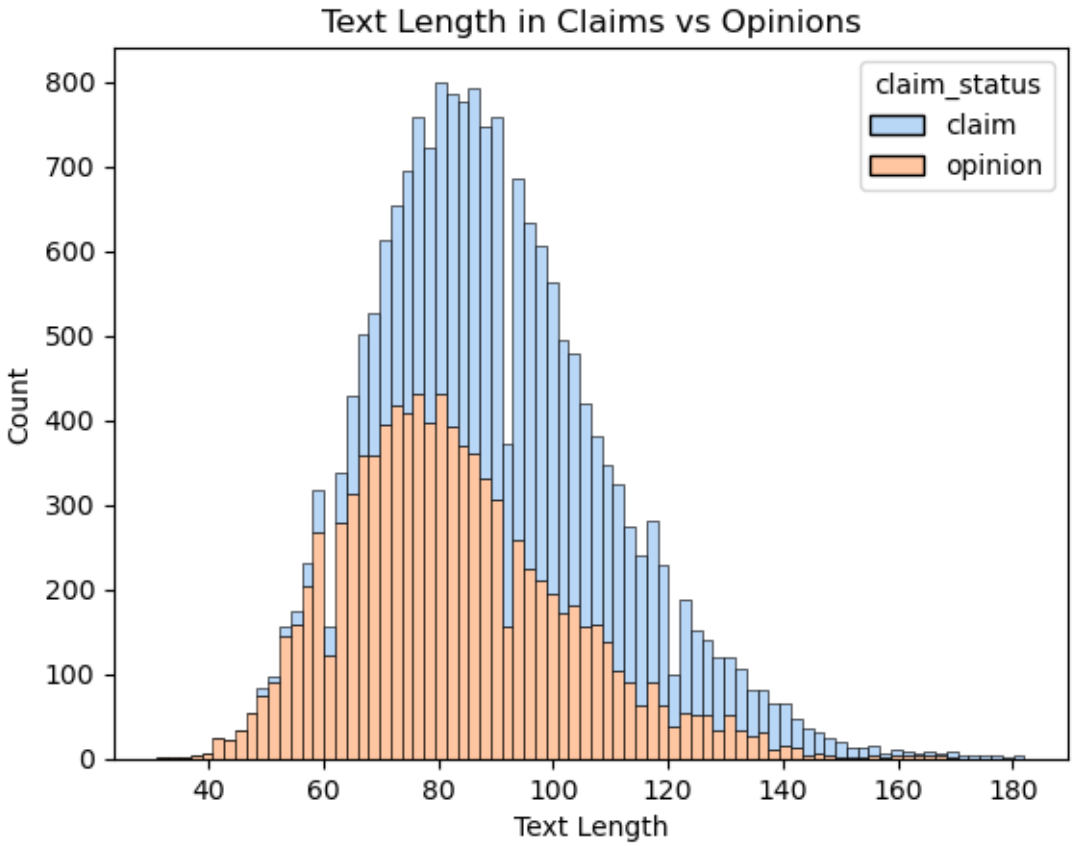
	#	claim_status	video_id	video_duration_sec	video_transcription_text	verified_status	author_ban_status	video_view_count	video_like_count	video_share_count	video_download_count	video_comment_count	text_length
0	1	claim	7017666017	59	someone shared with me that drone deliveries a…	not verified	under review	343296.0	19425.0	241.0	1.0	0.0	97
1	2	claim	4014381136	32	someone shared with me that there are more mic…	not verified	active	140877.0	77355.0	19034.0	1161.0	684.0	107
2	3	claim	9859838091	31	someone shared with me that american industria…	not verified	active	902185.0	97690.0	2858.0	833.0	329.0	137
3	4	claim	1866847991	25	someone shared with me that the metro of st. p…	not verified	active	437506.0	239954.0	34812.0	1234.0	584.0	131
4	5	claim	7105231098	19	someone shared with me that the number of busi…	not verified	active	56167.0	34987.0	4110.0	547.0	152.0	128

[16]:
```
# Calculate the average text_length for claims and opinions.
df[['claim_status','text_length']].groupby(['claim_status']).mean()
```

[16]:

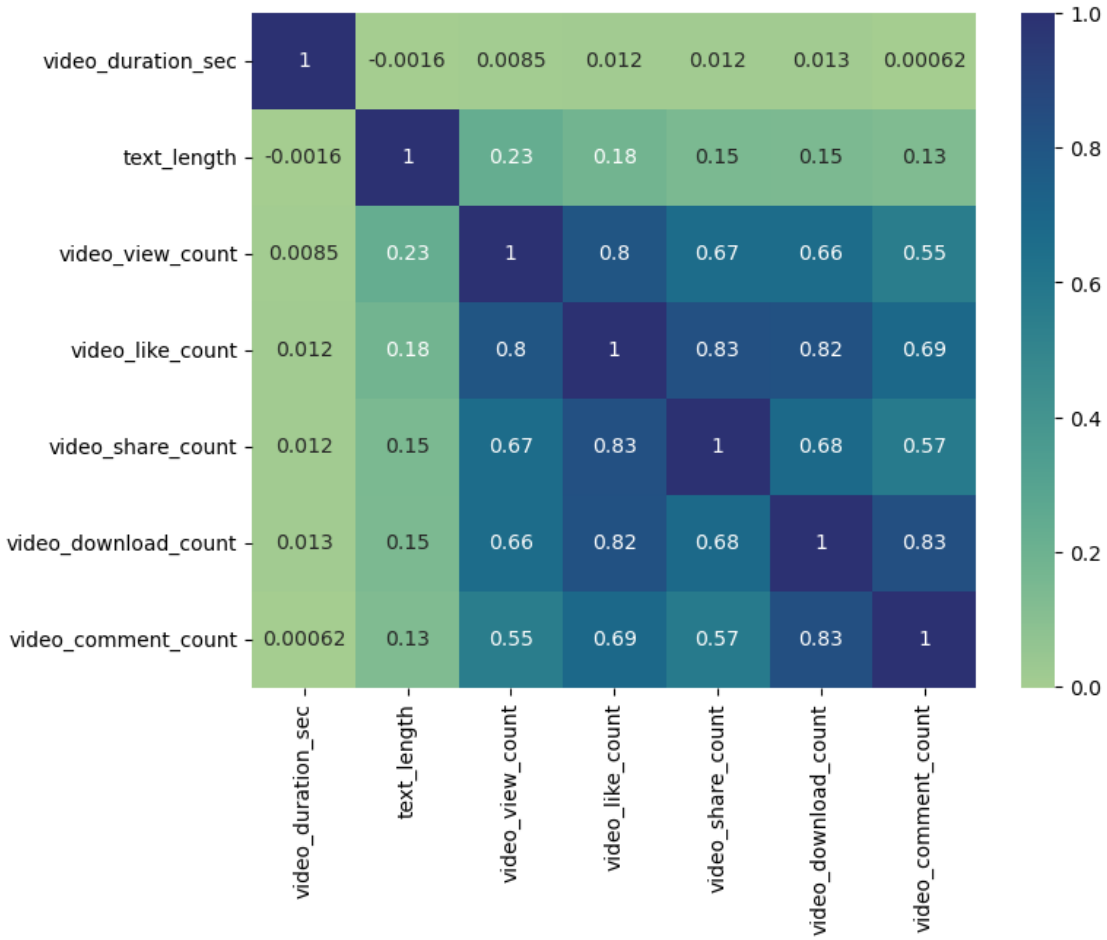
claim_status	text_length
claim	95.376978
opinion	82.722562

[17]:
```
# Visualize the distribution of `video_transcription_text` length for claims␣
 ↪and opinions
# Create two histograms in one plot
sb.histplot(data=df, stat="count", multiple="stack", x="text_length",␣
 ↪kde=False, palette="pastel", hue="claim_status", element="bars", legend=True)
plt.title("Text Length in Claims vs Opinions")
plt.xlabel("Text Length")
plt.ylabel("Count")
plt.show()
```



[18]:
```
# Create a heatmap to visualize how correlated variables are
plt.figure(figsize=(8,6))
sb.
 ↪heatmap(df[['video_duration_sec','text_length','video_view_count','video_like_count','video
 ↪corr(numeric_only=True),annot=True, cmap="crest")
# claim_status, verified_status, and author_ban_status are not numeric so they␣
 ↪were excluded
```

[18]: <Axes: >



Feature selection and transformation

[19]:
```
encode = df.copy()
# Encode target and catgorical variables.
encode = pd.get_dummies(encode, columns = ['author_ban_status'], dtype=float)
encode['claim_status'] = encode['claim_status'].replace({'opinion': 0, 'claim':␣
 ↪1})
```

```
encode['verified_status'] = encode['verified_status'].replace({'not verified':␣
↪0, 'verified': 1})
encode = encode.drop(["video_transcription_text","video_id"], axis=1)
# Display first few rows
encode.head(5)
```

[19]:

	#	claim_status	video_duration_sec	verified_status	video_view_count	video_like_count	video_share_count	video_download_count	video_comment_count	text_length	author_ban_status_active	author_ban_status_banned	author_ban_status_under review
0	1	1	59	0	343296.0	19425.0	241.0	1.0	0.0	97	0.0	0.0	1.0
1	2	1	32	0	140877.0	77355.0	19034.0	1161.0	684.0	107	1.0	0.0	0.0
2	3	1	31	0	902185.0	97690.0	2858.0	833.0	329.0	137	1.0	0.0	0.0
3	4	1	25	0	437506.0	239954.0	34812.0	1234.0	584.0	131	1.0	0.0	0.0
4	5	1	19	0	56167.0	34987.0	4110.0	547.0	152.0	128	1.0	0.0	0.0

2.0.3 Split the data

[20]:
```
# Assign target variable.
y = encode['claim_status']
```

[21]:
```
#Isolate features
X =␣
↪encode[['video_duration_sec','verified_status','text_length','video_view_count','video_like
↪review']]

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

[21]:

	video_duration_sec	verified_status	text_length	video_view_count	video_like_count	video_share_count	video_download_count	video_comment_count	author_ban_status_active	author_ban_status_banned	author_ban_status_under review
0	59	0	97	343296.0	19425.0	241.0	1.0	0.0	0.0	0.0	1.0
1	32	0	107	140877.0	77355.0	19034.0	1161.0	684.0	1.0	0.0	0.0
2	31	0	137	902185.0	97690.0	2858.0	833.0	329.0	1.0	0.0	0.0
3	25	0	131	437506.0	239954.0	34812.0	1234.0	584.0	1.0	0.0	0.0
4	19	0	128	56167.0	34987.0	4110.0	547.0	152.0	1.0	0.0	0.0

Create train/validate/test sets

[22]:
```
# Split data into training and testing sets, 80/20.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[23]:
```
# Split 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.
X_train, X_validation, y_train, y_validation = train_test_split(X_train,␣
↪y_train, test_size=0.25, random_state=42)
```

[24]:
```
# Confirm that the dimensions of the training, validation, and testing sets are␣
↪in alignment.
print(
np.shape(X_train),
np.shape(X_test),
np.shape(X_validation),
np.shape(y_train),
np.shape(y_test),
```

```
np.shape(y_validation))
```

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

2.0.4 Build models

2.0.5 Build a random forest model

Fit a random forest model to the training set. Use cross-validation to tune the hyperparameters and select the model that performs best on recall.

[25]:
```
# Instantiate the random forest classifier
rf = RandomForestClassifier(random_state=0)

# Create a dictionary of hyperparameters to tune
cv_params = {'max_depth': [5, 7, None],
             'max_features': [0.3, 0.6],
            #  'max_features': 'auto'
             'max_samples': [0.7],
             'min_samples_leaf': [1,2],
             'min_samples_split': [2,3],
             'n_estimators': [75,100,200],
             }

# Define a dictionary of scoring metrics to capture
scoring = {'accuracy', 'precision', 'recall', 'f1'}

# Instantiate the GridSearchCV object
rf_cv = GridSearchCV(rf, cv_params, scoring=scoring, cv=5, refit='recall')
```

[26]:
```
rf_cv.fit(X_train, y_train)
```

[26]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(random_state=0),
             param_grid={'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]},
             refit='recall', scoring={'accuracy', 'precision', 'f1', 'recall'})
```

[27]:
```
# Isolate the row of the df with the max(mean precision score)
rf_cv_results = pd.DataFrame(rf_cv.cv_results_)
rf_cv_highest_mean_recall = rf_cv_results['mean_test_recall'].idxmax()
#hmr will be used for highest mean recall going forward

# Extract the hyperparameters and other relevant information from that row
rf_cv_hmr_params = rf_cv_results.loc[rf_cv_highest_mean_recall, 'params']
rf_cv_hmr = rf_cv_results.loc[rf_cv_highest_mean_recall, 'mean_test_recall']
```

```
print("Best Hyperparameters:", rf_cv_hmr_params)
print("Best Mean Recall Score:", rf_cv_hmr)
```

```
Best Hyperparameters: {'max_depth': 5, 'max_features': 0.6, 'max_samples': 0.7,
'min_samples_leaf': 1, 'min_samples_split': 3, 'n_estimators': 100}
Best Mean Recall Score: 0.9913766326500252
```

[28]:
```
  # Get all the results from the CV and put them in a df
rf_cv = GridSearchCV(rf, cv_params, scoring=scoring, cv=5, refit='precision')
rf_cv.fit(X,y)

rf_cv_results = pd.DataFrame(rf_cv.cv_results_)

rf_cv_results.head(5)
```

[28]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	\
0	0.473377	0.006753	0.027958	0.001274	
1	0.628722	0.005648	0.033362	0.000294	
2	1.256286	0.011271	0.058195	0.000896	
3	0.470287	0.001684	0.026950	0.000468	
4	0.623750	0.004230	0.033466	0.000495	

	param_max_depth	param_max_features	param_max_samples	param_min_samples_leaf	\
0	5	0.3	0.7	1	
1	5	0.3	0.7	1	
2	5	0.3	0.7	1	
3	5	0.3	0.7	1	
4	5	0.3	0.7	1	

	param_min_samples_split	param_n_estimators	…	std_test_f1	rank_test_f1	\
0	2	75	…	0.001266	59	
1	2	100	…	0.001266	59	
2	2	200	…	0.001356	44	
3	3	75	…	0.001266	59	
4	3	100	…	0.001181	51	

	split0_test_recall	split1_test_recall	split2_test_recall	\
0	0.992716	0.991675	0.986472	
1	0.992716	0.991675	0.986472	
2	0.992716	0.991675	0.986472	
3	0.992716	0.991675	0.986472	
4	0.992716	0.991675	0.986472	

	split3_test_recall	split4_test_recall	mean_test_recall	std_test_recall	\
0	0.988548	0.992712	0.990425	0.002498	
1	0.988548	0.992712	0.990425	0.002498	
2	0.988548	0.993233	0.990529	0.002600	

```
3            0.988548            0.992712           0.990425          0.002498
4            0.988548            0.992712           0.990425          0.002498

   rank_test_recall
0                64
1                64
2                61
3                64
4                64

[5 rows x 43 columns]
```

[29]:
```
  # Isolate the row of the df with the max(mean precision score)
rf_cv_highest_mean_precision = rf_cv_results['mean_test_precision'].idxmax()
#hmp will be used for highest mean precision going forward

# Extract the hyperparameters and other relevant information from that row
rf_cv_hmp_params = rf_cv_results.loc[rf_cv_highest_mean_precision, 'params']
rf_cv_hmp = rf_cv_results.loc[rf_cv_highest_mean_precision,␣
 ↪'mean_test_precision']

print("Best Hyperparameters:", rf_cv_hmp_params)
print("Best Mean Precision Score:", rf_cv_hmp)
```

```
Best Hyperparameters: {'max_depth': 5, 'max_features': 0.6, 'max_samples': 0.7,
'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
Best Mean Precision Score: 1.0
```

[30]:
```
  # Get all the results from the CV and put them in a df
rf_cv = GridSearchCV(rf, cv_params, scoring=scoring, cv=5, refit='accuracy')
rf_cv.fit(X,y)

rf_cv_results = pd.DataFrame(rf_cv.cv_results_)

rf_cv_results.head(5)
```

[30]:
```
   mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
0       0.505742      0.003951         0.029054        0.003331
1       0.692256      0.012236         0.038147        0.001519
2       1.365793      0.007416         0.070656        0.014096
3       0.504690      0.003853         0.028368        0.002216
4       0.705372      0.020849         0.036725        0.003364

  param_max_depth param_max_features param_max_samples param_min_samples_leaf  \
0               5               0.3               0.7                      1
1               5               0.3               0.7                      1
2               5               0.3               0.7                      1
```

```
3                5                0.3                0.7                       1
4                5                0.3                0.7                       1

  param_min_samples_split param_n_estimators  … std_test_f1  rank_test_f1  \
0                       2                 75  …    0.001266            59
1                       2                100  …    0.001266            59
2                       2                200  …    0.001356            44
3                       3                 75  …    0.001266            59
4                       3                100  …    0.001181            51

   split0_test_recall  split1_test_recall  split2_test_recall  \
0            0.992716            0.991675            0.986472
1            0.992716            0.991675            0.986472
2            0.992716            0.991675            0.986472
3            0.992716            0.991675            0.986472
4            0.992716            0.991675            0.986472

   split3_test_recall  split4_test_recall  mean_test_recall  std_test_recall  \
0            0.988548            0.992712          0.990425         0.002498
1            0.988548            0.992712          0.990425         0.002498
2            0.988548            0.993233          0.990529         0.002600
3            0.988548            0.992712          0.990425         0.002498
4            0.988548            0.992712          0.990425         0.002498

   rank_test_recall
0                64
1                64
2                61
3                64
4                64

[5 rows x 43 columns]
```

[31]:
```python
  # Isolate the row of the df with the max(mean precision score)
rf_cv_highest_mean_accuracy = rf_cv_results['mean_test_accuracy'].idxmax()
#hma will be used for highest mean accuracy going forward

# Extract the hyperparameters and other relevant information from that row
rf_cv_hma_params = rf_cv_results.loc[rf_cv_highest_mean_accuracy, 'params']
rf_cv_hma = rf_cv_results.loc[rf_cv_highest_mean_accuracy, 'mean_test_accuracy']

print("Best Hyperparameters:", rf_cv_hma_params)
print("Best Mean Accuracy Score:", rf_cv_hma)
```

```
Best Hyperparameters: {'max_depth': None, 'max_features': 0.6, 'max_samples':
0.7, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
Best Mean Accuracy Score: 0.9954936785614834
```

[32]:
```
  # Get all the results from the CV and put them in a df
rf_cv = GridSearchCV(rf, cv_params, scoring=scoring, cv=5, refit='f1')
rf_cv.fit(X,y)

rf_cv_results = pd.DataFrame(rf_cv.cv_results_)

rf_cv_results.head(5)
```

[32]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_max_features	param_max_samples	param_min_samples_leaf	param_min_samples_split	param_n_estimators	…	std_test_f1	rank_test_f1	split0_test_recall	split1_test_recall	split2_test_recall	split3_test_recall	split4_test_recall	mean_test_recall	std_test_recall	rank_test_recall
0	0.495156	0.003947	0.027613	0.000997	5	0.3	0.7	1	2	75	…	0.001266	59	0.992716	0.991675	0.986472	0.988548	0.992712	0.990425	0.002498	64
1	0.650240	0.005396	0.034940	0.001098	5	0.3	0.7	1	2	100	…	0.001266	59	0.992716	0.991675	0.986472	0.988548	0.992712	0.990425	0.002498	64
2	1.324996	0.027641	0.062113	0.001610	5	0.3	0.7	1	2	200	…	0.001356	44	0.992716	0.991675	0.986472	0.988548	0.993233	0.990529	0.002600	61
3	0.493739	0.003765	0.027646	0.001272	5	0.3	0.7	1	3	75	…	0.001266	59	0.992716	0.991675	0.986472	0.988548	0.992712	0.990425	0.002498	64
4	0.646173	0.003126	0.033779	0.001010	5	0.3	0.7	1	3	100	…	0.001181	51	0.992716	0.991675	0.986472	0.988548	0.992712	0.990425	0.002498	64

```
3                64
4                64

[5 rows x 43 columns]
```

[33]:
```
# Isolate the row of the df with the max(mean precision score)
rf_cv_highest_mean_f1 = rf_cv_results['mean_test_f1'].idxmax()
#hmf will be used for highest mean f1 going forward

# Extract the hyperparameters and other relevant information from that row
rf_cv_hmf_params = rf_cv_results.loc[rf_cv_highest_mean_f1, 'params']
rf_cv_hmf = rf_cv_results.loc[rf_cv_highest_mean_f1, 'mean_test_f1']

print("Best Hyperparameters:", rf_cv_hmf_params)
print("Best Mean f1 Score:", rf_cv_hmf)
```

```
Best Hyperparameters: {'max_depth': None, 'max_features': 0.6, 'max_samples':
0.7, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
Best Mean f1 Score: 0.9955038085928759
```

How well did the model perform?

This model performs very well. Its best recall score is 99.1%, its best precision score is 100%, its best accuracy score is 99.5%, and its best f1 score is 99.5%.

2.0.6 Build an XGBoost model

[34]:
```
# Instantiate the XGBoost classifier
XGB = XGBClassifier(objective='binary:logistic', random_state=0)

# Create a dictionary of hyperparameters to tune
cv_params = {'max_depth': [3, 10, None],
             'learning_rate': [0.01, 0.1, 0.2, 0.3],
             'min_child_weight': [1, 10, 100],
             'n_estimators': [10,500,1000],
             }

# Define a dictionary of scoring metrics to capture
scoring = {'accuracy', 'precision', 'recall', 'f1'}
```

[35]:
```
# Get all the results from the CV and put them in a df
XGB_cv = GridSearchCV(XGB, cv_params, scoring=scoring, cv=5, refit='accuracy')
XGB_cv.fit(X,y)

XGB_cv_results = pd.DataFrame(XGB_cv.cv_results_)

XGB_cv_results.head(5)
```

[35]:
```
   mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
0       0.052668      0.004455         0.009008        0.000247
1       1.957019      0.046079         0.017376        0.001352
2       3.896254      0.071652         0.026200        0.002024
3       0.049149      0.002624         0.009341        0.000815
4       1.877974      0.039779         0.016065        0.000225

  param_learning_rate param_max_depth param_min_child_weight  \
0                0.01               3                      1
1                0.01               3                      1
2                0.01               3                      1
3                0.01               3                     10
4                0.01               3                     10

  param_n_estimators                                             params  \
0                 10  {'learning_rate': 0.01, 'max_depth': 3, 'min_c…
1                500  {'learning_rate': 0.01, 'max_depth': 3, 'min_c…
2               1000  {'learning_rate': 0.01, 'max_depth': 3, 'min_c…
3                 10  {'learning_rate': 0.01, 'max_depth': 3, 'min_c…
4                500  {'learning_rate': 0.01, 'max_depth': 3, 'min_c…

   split0_test_accuracy  …  std_test_f1  rank_test_f1  split0_test_recall  \
0              0.996332  …     0.001304            11            0.992716
1              0.996332  …     0.001334            16            0.992716
2              0.996332  …     0.001370            21            0.992716
3              0.996070  …     0.001215            40            0.992196
4              0.996070  …     0.001215            40            0.992196

   split1_test_recall  split2_test_recall  split3_test_recall  \
0            0.992196            0.986472            0.989589
1            0.991675            0.986472            0.989068
2            0.991675            0.986472            0.989068
3            0.990635            0.986472            0.988027
4            0.990635            0.986472            0.988027

   split4_test_recall  mean_test_recall  std_test_recall  rank_test_recall
0            0.993753          0.990945         0.002624                15
1            0.993753          0.990737         0.002641                34
2            0.993753          0.990737         0.002641                34
3            0.992712          0.990008         0.002405                58
4            0.992712          0.990008         0.002405                58

[5 rows x 41 columns]
```

[36]:
```
# Isolate the row of the df with the max(mean accuracy score)
XGB_cv_highest_mean_accuracy = XGB_cv_results['mean_test_accuracy'].idxmax()
#hma will be used for highest mean accuracy going forward
```

```
# Extract the hyperparameters and other relevant information from that row
XGB_cv_hma_params = XGB_cv_results.loc[XGB_cv_highest_mean_accuracy, 'params']
XGB_cv_hma = XGB_cv_results.loc[XGB_cv_highest_mean_accuracy,␣
 ↪'mean_test_accuracy']

print("Best Hyperparameters:", XGB_cv_hma_params)
print("Best Mean Accuracy Score:", XGB_cv_hma)
```

```
Best Hyperparameters: {'learning_rate': 0.1, 'max_depth': None,
'min_child_weight': 1, 'n_estimators': 10}
Best Mean Accuracy Score: 0.9955460757320361
```

[37]:
```
# Get all the results from the CV and put them in a df
XGB_cv = GridSearchCV(XGB, cv_params, scoring=scoring, cv=5, refit='precision')
XGB_cv.fit(X,y)

XGB_cv_results = pd.DataFrame(XGB_cv.cv_results_)

XGB_cv_results.head(5)
```

[37]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	\
0	0.052115	0.006647	0.010141	0.001831	
1	1.979574	0.046984	0.016584	0.000741	
2	3.907530	0.050503	0.024629	0.000894	
3	0.049489	0.002635	0.009049	0.000164	
4	1.888487	0.055996	0.017963	0.002509	

	param_learning_rate	param_max_depth	param_min_child_weight	\
0	0.01	3	1	
1	0.01	3	1	
2	0.01	3	1	
3	0.01	3	10	
4	0.01	3	10	

	param_n_estimators	params	\
0	10	{'learning_rate': 0.01, 'max_depth': 3, 'min_c…	
1	500	{'learning_rate': 0.01, 'max_depth': 3, 'min_c…	
2	1000	{'learning_rate': 0.01, 'max_depth': 3, 'min_c…	
3	10	{'learning_rate': 0.01, 'max_depth': 3, 'min_c…	
4	500	{'learning_rate': 0.01, 'max_depth': 3, 'min_c…	

	split0_test_accuracy	…	std_test_f1	rank_test_f1	split0_test_recall	\
0	0.996332	…	0.001304	11	0.992716	
1	0.996332	…	0.001334	16	0.992716	
2	0.996332	…	0.001370	21	0.992716	
3	0.996070	…	0.001215	40	0.992196	

```
4              0.996070  …      0.001215            40            0.992196

   split1_test_recall  split2_test_recall  split3_test_recall  \
0            0.992196            0.986472            0.989589
1            0.991675            0.986472            0.989068
2            0.991675            0.986472            0.989068
3            0.990635            0.986472            0.988027
4            0.990635            0.986472            0.988027

   split4_test_recall  mean_test_recall  std_test_recall  rank_test_recall
0            0.993753          0.990945         0.002624                15
1            0.993753          0.990737         0.002641                34
2            0.993753          0.990737         0.002641                34
3            0.992712          0.990008         0.002405                58
4            0.992712          0.990008         0.002405                58

[5 rows x 41 columns]
```

[38]:
```
# Isolate the row of the df with the max(mean precision score)
XGB_cv_highest_mean_precision = XGB_cv_results['mean_test_precision'].idxmax()
#hmp will be used for highest mean precision going forward

# Extract the hyperparameters and other relevant information from that row
XGB_cv_hmp_params = XGB_cv_results.loc[XGB_cv_highest_mean_precision, 'params']
XGB_cv_hmp = XGB_cv_results.loc[XGB_cv_highest_mean_precision,␣
 ↪'mean_test_precision']

print("Best Hyperparameters:", XGB_cv_hmp_params)
print("Best Mean Precision Score:", XGB_cv_hmp)
```

```
Best Hyperparameters: {'learning_rate': 0.01, 'max_depth': 3,
'min_child_weight': 1, 'n_estimators': 500}
Best Mean Precision Score: 1.0
```

[39]:
```
# Get all the results from the CV and put them in a df
XGB_cv = GridSearchCV(XGB, cv_params, scoring=scoring, cv=5, refit='recall')
XGB_cv.fit(X,y)

XGB_cv_results = pd.DataFrame(XGB_cv.cv_results_)

XGB_cv_results.head(5)
```

[39]:
```
   mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
0       0.056480      0.004745         0.009525        0.000710
1       2.216857      0.053061         0.019781        0.003713
2       4.464121      0.051480         0.027998        0.004693
3       0.055165      0.006054         0.011286        0.002894
```

```
4       2.164570      0.025310         0.018582        0.003628

  param_learning_rate param_max_depth param_min_child_weight  \
0                0.01               3                      1
1                0.01               3                      1
2                0.01               3                      1
3                0.01               3                     10
4                0.01               3                     10

  param_n_estimators                                             params  \
0                 10  {'learning_rate': 0.01, 'max_depth': 3, 'min_c…
1                500  {'learning_rate': 0.01, 'max_depth': 3, 'min_c…
2               1000  {'learning_rate': 0.01, 'max_depth': 3, 'min_c…
3                 10  {'learning_rate': 0.01, 'max_depth': 3, 'min_c…
4                500  {'learning_rate': 0.01, 'max_depth': 3, 'min_c…

   split0_test_accuracy  …  std_test_f1  rank_test_f1  split0_test_recall  \
0              0.996332  …     0.001304            11            0.992716
1              0.996332  …     0.001334            16            0.992716
2              0.996332  …     0.001370            21            0.992716
3              0.996070  …     0.001215            40            0.992196
4              0.996070  …     0.001215            40            0.992196

   split1_test_recall  split2_test_recall  split3_test_recall  \
0            0.992196            0.986472            0.989589
1            0.991675            0.986472            0.989068
2            0.991675            0.986472            0.989068
3            0.990635            0.986472            0.988027
4            0.990635            0.986472            0.988027

   split4_test_recall  mean_test_recall  std_test_recall  rank_test_recall
0            0.993753          0.990945         0.002624                15
1            0.993753          0.990737         0.002641                34
2            0.993753          0.990737         0.002641                34
3            0.992712          0.990008         0.002405                58
4            0.992712          0.990008         0.002405                58

[5 rows x 41 columns]
```

[40]:
```python
# Isolate the row of the df with the max(mean recall score)
XGB_cv_highest_mean_recall = XGB_cv_results['mean_test_recall'].idxmax()
#hmr will be used for highest mean recall going forward

# Extract the hyperparameters and other relevant information from that row
XGB_cv_hmr_params = XGB_cv_results.loc[XGB_cv_highest_mean_recall, 'params']
XGB_cv_hmr = XGB_cv_results.loc[XGB_cv_highest_mean_recall, 'mean_test_recall']
```

```
print("Best Hyperparameters:", XGB_cv_hmr_params)
print("Best Mean Recall Score:", XGB_cv_hmr)
```

```
Best Hyperparameters: {'learning_rate': 0.01, 'max_depth': 10,
'min_child_weight': 1, 'n_estimators': 10}
Best Mean Recall Score: 0.9913614841385616
```

[41]:
```
# Get all the results from the CV and put them in a df
XGB_cv = GridSearchCV(XGB, cv_params, scoring=scoring, cv=5, refit='f1')
XGB_cv.fit(X,y)

XGB_cv_results = pd.DataFrame(XGB_cv.cv_results_)

XGB_cv_results.head(5)
```

[41]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	\
0	0.049175	0.002202	0.010131	0.002313	
1	1.961853	0.059946	0.016457	0.000319	
2	3.910030	0.128610	0.025245	0.002675	
3	0.048863	0.003118	0.009364	0.001048	
4	1.870303	0.037521	0.016613	0.001837	

	param_learning_rate	param_max_depth	param_min_child_weight	\
0	0.01	3	1	
1	0.01	3	1	
2	0.01	3	1	
3	0.01	3	10	
4	0.01	3	10	

	param_n_estimators	params	\
0	10	{'learning_rate': 0.01, 'max_depth': 3, 'min_c…	
1	500	{'learning_rate': 0.01, 'max_depth': 3, 'min_c…	
2	1000	{'learning_rate': 0.01, 'max_depth': 3, 'min_c…	
3	10	{'learning_rate': 0.01, 'max_depth': 3, 'min_c…	
4	500	{'learning_rate': 0.01, 'max_depth': 3, 'min_c…	

	split0_test_accuracy	…	std_test_f1	rank_test_f1	split0_test_recall	\
0	0.996332	…	0.001304	11	0.992716	
1	0.996332	…	0.001334	16	0.992716	
2	0.996332	…	0.001370	21	0.992716	
3	0.996070	…	0.001215	40	0.992196	
4	0.996070	…	0.001215	40	0.992196	

	split1_test_recall	split2_test_recall	split3_test_recall	\
0	0.992196	0.986472	0.989589	
1	0.991675	0.986472	0.989068	
2	0.991675	0.986472	0.989068	

```
3            0.990635            0.986472            0.988027
4            0.990635            0.986472            0.988027

   split4_test_recall  mean_test_recall  std_test_recall  rank_test_recall
0            0.993753          0.990945         0.002624                15
1            0.993753          0.990737         0.002641                34
2            0.993753          0.990737         0.002641                34
3            0.992712          0.990008         0.002405                58
4            0.992712          0.990008         0.002405                58

[5 rows x 41 columns]
```

[43]:
```
# Isolate the row of the df with the max(mean f1 score)
XGB_cv_highest_mean_f1 = XGB_cv_results['mean_test_f1'].idxmax()
#hmf will be used for highest mean f1 going forward

# Extract the hyperparameters and other relevant information from that row
XGB_cv_hmf_params = XGB_cv_results.loc[XGB_cv_highest_mean_f1, 'params']
XGB_cv_hmf = XGB_cv_results.loc[XGB_cv_highest_mean_f1, 'mean_test_f1']

print("Best Hyperparameters:", XGB_cv_hmf_params)
print("Best Mean f1 Score:", XGB_cv_hmf)
```

```
Best Hyperparameters: {'learning_rate': 0.1, 'max_depth': None,
'min_child_weight': 1, 'n_estimators': 10}
Best Mean f1 Score: 0.9955553840532527
```

How well did the model perform?

This model performs very well. Its best recall score is 99.1%, its best precision score is 100%, its best accuracy score is 99.6%, and its best f1 score is 99.6%.

2.0.7 Evaluate model

Random forest

[44]:
```
# Use the random forest "best estimator" model to get predictions on the␣
  ↪encoded testing set
rf_predict = rf_cv.best_estimator_.predict(X_validation)
```

[45]:
```
# Display the predictions on the encoded testing set
rf_predict
```

[45]:
```
array([1, 0, 0, …, 0, 0, 1])
```

[48]:
```
# Display the true labels of the testing set
y_validation
```

[48]:
```
1871     1
16574    0
17741    0
17214    0
17821    0
        ..
15929    0
12177    0
18295    0
17339    0
9186     1
Name: claim_status, Length: 3817, dtype: int64
```

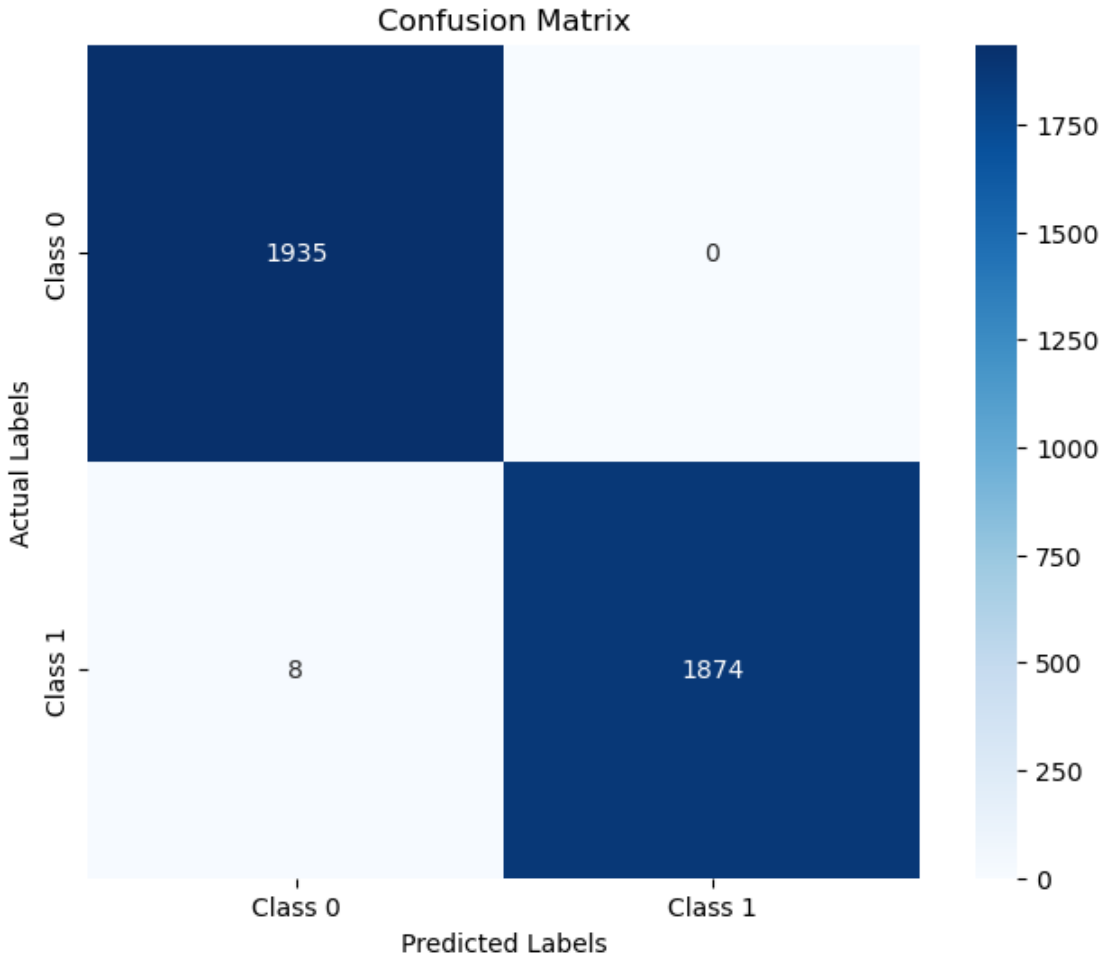
[50]:
```
# Create a confusion matrix to visualize the results of the classification model
# Compute values for confusion matrix
rf_cm = confusion_matrix(y_validation,rf_predict)

# Create display of confusion matrix
rf_cm_display = ConfusionMatrixDisplay(confusion_matrix=rf_cm)

# Show the heatmap
plt.figure(figsize=(8, 6))
sb.heatmap(rf_cm, annot=True, fmt="d", cmap="Blues", square=True,
            xticklabels=['Class 0', 'Class 1'],
            yticklabels=['Class 0', 'Class 1'])

# Add labels and title
plt.xlabel('Predicted Labels')
plt.ylabel('Actual Labels')
plt.title('Confusion Matrix')

# Show the heatmap
plt.show()
```



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

[62]:
```
# Create a classification report
# Create classification report for random forest model
rf_cv_report = classification_report(y_validation,rf_predict)
print(rf_cv_report)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1935
           1       1.00      1.00      1.00      1882

    accuracy                           1.00      3817
   macro avg       1.00      1.00      1.00      3817
weighted avg       1.00      1.00      1.00      3817
```

The confusion matrix and classification report show that this Random Forest models is almost 100% accurate.

XGBoost

[53]:
```
#Evaluate XGBoost model
XGB_predict = XGB_cv.best_estimator_.predict(X_validation)
```

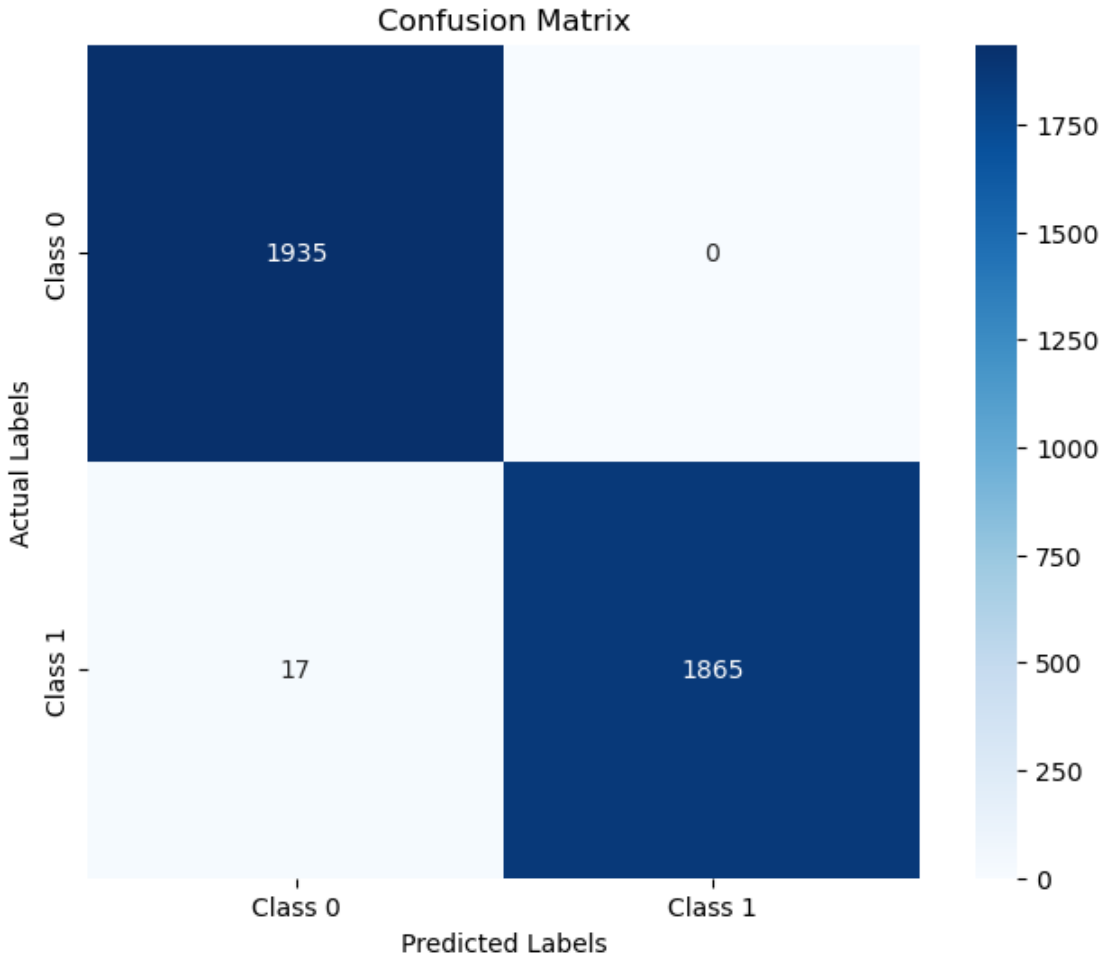
[55]:
```
# Compute values for confusion matrix
XGB_cm = confusion_matrix(y_validation,XGB_predict)

# Create display of confusion matrix
XGB_cm_display = ConfusionMatrixDisplay(confusion_matrix=XGB_cm)

# Show the heatmap
plt.figure(figsize=(8, 6))
sb.heatmap(XGB_cm, annot=True, fmt="d", cmap="Blues", square=True,
            xticklabels=['Class 0', 'Class 1'],
            yticklabels=['Class 0', 'Class 1'])

# Add labels and title
plt.xlabel('Predicted Labels')
plt.ylabel('Actual Labels')
plt.title('Confusion Matrix')

# Show the heatmap
plt.show()
```



[56]:
```
# Create a classification report
XGB_cv_report = classification_report(y_validation,rf_predict)
XGB_cv_report
```

[56]:
```
'              precision    recall  f1-score   support\n\n           0
1.00      1.00      1.00      1935\n           1       1.00      1.00      1.00
1882\n\n    accuracy                           1.00      3817\n   macro avg
1.00      1.00      1.00      3817\nweighted avg       1.00      1.00      1.00
3817\n'
```

XGBoost model and the Random Forest model have almost the same levels of accuracy. Either model would probably be acceptable, but the Random Forest model was slightly more effective so it will be used as the champion model.

2.0.8 Use champion model to predict on test data

[58]:
```
# champion model prediction based on test data
# the champion model is the random forest by a very small margin
champ_rf_predict = rf_cv.best_estimator_.predict(X_test)
```

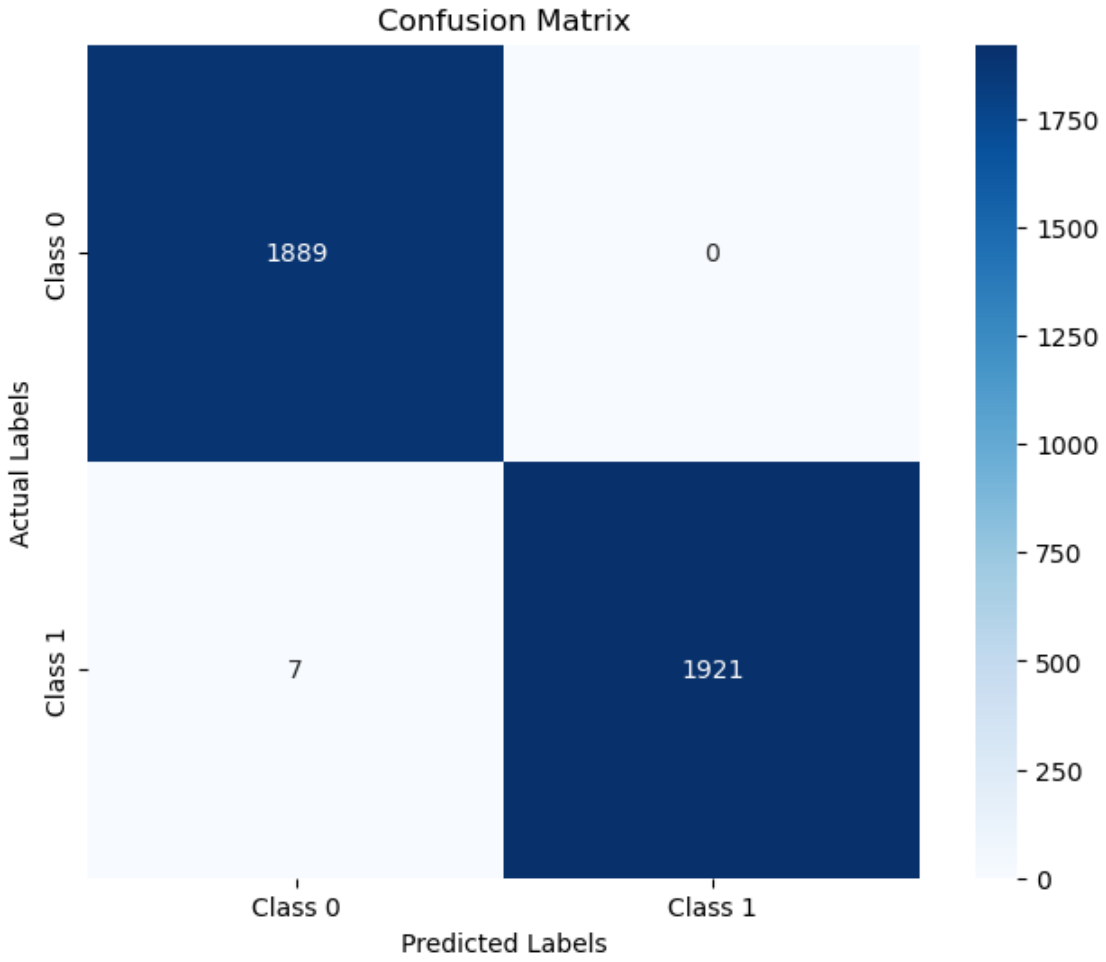
[60]:
```
# Compute values for confusion matrix
champ_rf_cm = confusion_matrix(y_test,champ_rf_predict)

# Create display of confusion matrix
champ_rf_cm_display = ConfusionMatrixDisplay(confusion_matrix=champ_rf_cm)

# Show the heatmap
plt.figure(figsize=(8, 6))
sb.heatmap(champ_rf_cm, annot=True, fmt="d", cmap="Blues", square=True,
            xticklabels=['Class 0', 'Class 1'],
            yticklabels=['Class 0', 'Class 1'])

# Add labels and title
plt.xlabel('Predicted Labels')
plt.ylabel('Actual Labels')
plt.title('Confusion Matrix')

# Show the heatmap
plt.show()
```



Feature importances of champion model

[61]:
```
# Display the feature importances of the champion model
importances = rf_cv.best_estimator_.feature_importances_

# Sort and display feature importances
feature_names = X_train.columns  # Replace with your feature names
sorted_feature_importances = sorted(zip(importances, feature_names),␣
 ↪reverse=True)
for importance, feature_name in sorted_feature_importances:
    print(f'{feature_name}: {importance:.4f}')
```

```
video_view_count: 0.6689
video_like_count: 0.2212
video_download_count: 0.0514
video_share_count: 0.0447
video_comment_count: 0.0099
```

```
text_length: 0.0017
video_duration_sec: 0.0016
author_ban_status_active: 0.0002
author_ban_status_banned: 0.0002
author_ban_status_under review: 0.0001
verified_status: 0.0000
```

What were the most predictive features? The engagement features representing views, likes, and downloads were the most impactful. Shares and comments significantly less so, and all other features contributed in little to no way. This indicates that to improve the model, more engagement features might be key.

2.0.9 Conclusion

1. This RandomForest model is very accuracy and can reliably be used to make predictions.
2. The chosen model is a RandomForest which samples the training dataset (60% of the available data) to create large numbers of decision trees and then selects the most likely result among all of those decision trees.
3. It is possible to take the transcription text and turn it into hundreds/thousands of ordered word pairs that are each their own column within the dataset. Using these additional features might improve model prediction accuracy.
4. The engagement features counted among the most impactful so finding more features that track engagement could improve the model. Some measurement options might be engagement feature per time increment, follower engagement vs nonfollower engagement, number of followers when the video was posted, and number of new followers.