Cal Capstone Project: Detect Comm

Module 20.1 Initial Report and Exploratory Data Analysis (EDA)

By Brad Brown

Goal

Using my previous research project and paper (title and link) as a starting point, can I improve on my initial research to achieve higher accuracy and more granular **detection of commercials within a television sports broadcast?**

The previous research project and paper was completed in March of 2024. The project had two stages. Stage 1 fed 15-second 'chunks' of text from 722 minutes of transcripts of the audio of TV sports broadcasts to ChatGPT via a python api, instructing it essentially to categorize each chunk as a commercial or a sports broadcast. In Stage 2, those answers along with other statistics about each chunk were then used to train a logistic regression model to try to improve the accuracy of the initial predictions.

The wider business goal of the Cal Capstone project is to improve the prediction accuracy overall. The Capstone project will be particularly focused on improving the Stage 2 accuracy:

1. Rework stage 1 and stage 2 model processing and metrics to **predict whether each sentence is a commercial or not**
2. **Use feature engineering** to improve accuracy and quality of the stage 2 model
3. Use the **XGBoost model rather than logisitc regression model**

See the section '*How we measure success*' below for detailed specific objectives.

Why does it matter?

There are potentially millions of people that would benefit from the ability to seamless detect commercials while watching sporting events. Sporting events are very popular but commercials are the opposite - very few people desire to see them. If the transitions to and from commercials can be detected reliably, then the possibility of providing interesting content to consumers during commercials via non-TV devices is feasible. For example, short inspirational or educational snippets of video could be triggered on a consumer's phone during a TV commercial. Therefore, every improvement possible to make the detection reliable matters to those consumers. Current techniques use visual image and audio signal processing modeling techniques. This project represents one of the first times large language models and other ML modeling techniques on that LLM's output are used to solve

the commercial detection problem. It has promise to be a more real-time practical solution or potentially be part of a more effective and efficient ensemble of modeling approach.

With the initial research project, raw accuracy of individual chunk predictions improved from Stage 1 (ChatGPT predictions) 87% to 91% in Stage 2 (Logistic Regression predictions using stage 1 outcomes as input). A 4% improvement is good, however, 15-second chunk granularity reduces its potential for practical use. Consumers care about accuracy around these transition points from mainly primary content to mainly commercial content and vice versa. If detection of a chunk is wrong, it wastes 15-seconds of consumer attention. Wrong twice in a row and it wastes 30-seconds - this result could mean the entire solution gets rejected by end users.

Therefore, **this project moves to a more granular sentence-based approach rather than a '15-second chunk' based approach.** While sentence block prediction is a more challenging task, if successful, it makes it a more useful result, especially if by using feature engineering and XGBoost, the system can produce an end result that is still 91% accuracy or higher.

How to measure success

The previous project correctly relied on Confusion Matrices as well as F1, Precision, and Recall accuracy metrics to judge commercial detection effectiveness. We will again use these measures but there are changes being made to the data structures and our analysis of the business needs also requires a change in our measurement approach.

Data structure change:Sentence-based:

- If the project de-emphasized the contiguous 15-second chunks of text and instead relied on single sentences, would the Stage 1 CM and F1 accuracy worsen, stay the same, or improve?

Stage 2 modeling changes: XGBoost and feature engineering:

- Can we train an XGBoost Decision Tree model on the LLM answers and metadata about each sentence to improve the accuracy more than the previous use of a Logistic Regression model?
- Can we use feature engineering such as adding features that take the square of existing features to improve accuracy?
- If the model can rely on past history of correct labels (rather than Stage 1 estimates), will it help it predict current sentence label better?

Lastly, but importantly, apply metrics for our results across different configurations to enable system designers to align more closely with the business goals and improve the user experience:

1. **Windowed, rolling, or 'smoothed' predictions:** Adopting 'windowed' predictions in Stage 1, means three of the same prediction in a row are needed to change the current prediction. In a sentence-based model, will we experience 'flip flopping' where the incorrect categorizations of one sentence flips the user to the wrong context and then the next sentence flips them back and then the cycle repeats? If so, this cycle might be very annoying to end users. Are the errors that cause a flip-flop between corect and incorrect predictions sentence by sentence worse than a consistent set of incorrect predictions that are slower to transition to the correct answer after a shift to or from commercials? In essence is it better to have your errors grouped together within long chains of sentences? We will measure both ways. If each has close to the same accuracy, then the product designers can choose the experience that users prefer.

2. **Handling Embedded commercials:** Should we differentiate between 'embedded commercials' vs 'standard commercials'. For example, oftentimes a sports broadcaster will promote a product while still talking about the sporting event in the same sentence: 'Let's take a look at our Coca Cola top player of the game statistics...'. *For this project, we will NOT attempt to categorize 'embedded commercials'*.

In summary the metrics we will use are F1, Recall and Precision as well as Confusion Matrices. They will be applied in mulitiple contexts.

BASELINE:

-->A) Previous research baseline values

XGBoost model:

-->B) Stage 1 using new sentence-based approach

-->C) Stage 2 using new sentence-based approach WITHOUT feature engineering and WITHOUT Windowed Predictions

-->D) Stage 2 using new sentence-based approach WITH feature engineering but WITHOUT Windowed Predictions

-->E) Stage 2 using new sentence-based approach WITH feature engineering and WITH Windowed Predictions

-->F) Stage 2 like Stage E except we make the assumption that we know the actual (ground truth) category for the previous sentence

Logistic Regression model:

-->G) Stage 2 using new sentence-based approach WITH feature engineering and WITH Windowed Predictions

-->H) Stage 2 like Stage G except we make the assumption that we know the actual (ground truth) category for the previous sentence

Ways to Run This Notebook - Configurations By Model, Feature Engineering and Data Assumptions


COMM DETECT Model Process Flow

(Not included in the write-up: Optimizing down to various number of features. For this report, we worked with 10 features after trying 2,4,5,10,15,and 20 features)

How to get there

1. Understand and explore our data from stage 1 and its outputs. The outputs of stage 1 are the inputs to stage 2. Look for data quality errors and insights about potential feature engineering and prioritization hints.
2. We will decide a data split strategy to help us for feature engineering and the modeling process.
3. Engineer features based on feature creation tools and subject matter expertise. As an integral part of the modeling process we will try nonlinear variants of our input data and then use feature importance tools to narrow down to the most promising feature set.
4. With a set of features defined, we will tune the parameters of the XGBoost and the Logistic Regression models, searching for the best hyper-parameters of each to tune each model.
5. We will make a final model choice and analyze how it performs on our held back test set.

Note that this process will be repeated for the different metrics defined in the previous section. It's possible that different models or different hyper-parameters perform better when optimizing for a particular metric

Data Understanding and Exploration

Data ultimately comes from 7+ hours of video including commercials. Here is link to one now, if you want to review a few seconds of one.

One of 4 videos in data set: Warriors Vs Nets NBA Basketball Video

As part of my project, I had the 7 hours of video transcribed to raw text files. Here is a sample of the format. The bracket characters [] hold the time stamp relative to the start of the recording. Timestamps are in format [hour:minute:second.microsecond]

> **[00:35:30.286]**
>
> Nets lead by three after one.

> Shout out to formerly the starters, no dunks crew, and to Ian Eagle himself. The bird loves wedgies. A good start to this night here at Barclays. **[00:35:45.332]**

> Can't be a pop star? Mmm, doubt it.

> Chauncey!

> This is rough, but it's also finished.

> I know what you want to do. Yeah, don't touch... Please don't touch that. **[00:36:00.149]**

> Get ready, America.

> Good morning, with Dulcolax.

> Good, good, good morning. Yeah.

> Try Dulcolax Chewy Fruit Bites for fast and gentle constipation relief in as little as 30 minutes. Making **[00:36:15.716]**

In [1]:
```
INSTALL_ON = False
if INSTALL_ON:
    %pip install pandas
    %pip install seaborn matplotlib
    %pip install numpy
    %pip install scikit-learn
    %pip install featuretools
    %pip install xgboost

%pwd
```

Out[1]:
```
'C:\\Users\\bbfor\\OneDrive\\code\\cal-cap'
```

In [2]:
```
# All imports needed to run this notebook code
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn.calibration import LabelEncoder
from sklearn.pipeline import Pipeline
from sklearn.metrics import balanced_accuracy_score, confusion_matrix, recall_score
from sklearn.metrics import classification_report
from sklearn.model_selection import StratifiedKFold, cross_val_score, train_test_sp
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import learning_curve
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SequentialFeatureSelector
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
```

```
from sklearn.model_selection import TimeSeriesSplit
from sklearn.inspection import permutation_importance
import featuretools as ft
import xgboost as xgb
```

In [3]:
```
# Data sets as pandas dataframes (snapshot of the sqlite db used for stage 1 proces
df_chunks = pd.read_csv('./db_export/chunks.csv')
df_instruction_prompts = pd.read_csv('./db_export/instruction_prompts.csv')
#df_llm_predictions = pd.read_csv('./db_export/llm_predictions.csv')
df_llm_predictions = pd.read_csv('./db_export/updated_llm_predictions.csv')
df_sessions = pd.read_csv('./db_export/sessions.csv')
df_sources = pd.read_csv('./db_export/sources.csv')
df_transitions = pd.read_csv('./db_export/transitions.csv')
df_text_blocks = pd.read_csv('./db_export/text_blocks2.csv')
print('sentence count:')
print(df_text_blocks.shape[0])
df_text_block_stat = pd.read_csv('./db_export/text_block_stat.csv')
```

```
sentence count:
10754
```

NOTEBOOK CONFIGURATION AND SETTINGS

In [4]:
```
# Set up some configuration and standard constants used in this notebook

# THE PRIMARY SYSTEM CONTROL VARIABLES ARE RUN_MODE and AUTO_SELECT_FEAT

# Change these to explore different configurations

#RUN_MIDE       Model   Feature       Windowed       Know the Category
#                   Engineering   Prediction     of Previous Sentence
#-------------------------------------------------------------
#--> C      XGBoost     No                No             No
#-------------------------------------------------------------
#--> D      XGBoost     Yes               No             No
#-------------------------------------------------------------
#--> E      XGBoost     Yes               Yes            No
#-------------------------------------------------------------
#--> F      XGBoost     Yes               Yes            Yes
#-------------------------------------------------------------
#--> G      LR      Yes           Yes            No
#-------------------------------------------------------------
#--> H      LR      Yes           Yes            Yes

#####################################
# Change this value per above chart
RUN_MODE = 'E'
#####################################

#########################################################################
# Turn AUTO_SELECT_FEAT to True if you want SFS to run (slow)
AUTO_SELECT_FEAT = True
#########################################################################
# If choose False, it will select based on hard-coded features
# These feature lists are created by observing what SFS produced previously as top
```

```
# (This approach drastically improves run-time when iterating on other aspects of s


# THE REST OF THE SETTINGS ARE LESS LIKELY TO CHANGE

# SFS Feature selection params
DEF_NUM_FEATURES = 10
FEAT_SCORING_METRIC = 'f1'

# figure size in charts
FIGX= 10
FIGY = 6

# To make calculations repeatable, use this whenever you can
# R_S means RANDOM STATE parameter
R_S = 42

# data quality check (%)
NAN_TOLERANCE = 0.03

# Control exponential decay feature engineering
DEF_ALPHA = .001


# test split by game number
TEST_SET_GAME_SESSION = 0


DEF_BASE_COL_LIST_LLM = [ 'session_id', 'text_chunk_used',
       'sports_broadcasting_streak_llm', 'commercial_streak_llm',
       'c_blocks_so_far_llm', 'sb_blocks_so_far_llm',
       'num_blocks_since_sb_llm', 'num_blocks_since_c_llm',
       'windowed_resp_class_c', 'sequence_num','response_classification_c', 'ground


DEF_BASE_COL_LIST_GT = [ 'session_id', 'text_chunk_used',
       'sports_broadcasting_streak_llm', 'commercial_streak_llm',
       'c_blocks_so_far_llm', 'sb_blocks_so_far_llm',
       'num_blocks_since_sb_llm', 'num_blocks_since_c_llm',
        'lag_1_ground_truth_label_c',
       'sports_broadcasting_streak_lag_1_gt', 'commercial_streak_lag_1_gt',
       'c_blocks_so_far_lag_1_gt', 'sb_blocks_so_far_lag_1_gt',
       'num_blocks_since_sb_lag_1_gt', 'num_blocks_since_c_lag_1_gt',
        'windowed_resp_class_c', 'sequence_num','response_classification_c', 'groun
```

In [5]:

```
# Controls whether we have access to ground truth of previous prediction
match RUN_MODE:
    case 'C':
        USE_LAG_GT_COLS = False
    case 'D':
        USE_LAG_GT_COLS = False
    case 'E':
        USE_LAG_GT_COLS = False
    case 'F':
        USE_LAG_GT_COLS = True
    case 'G':
```

```
        USE_LAG_GT_COLS = False
    case 'H':
        USE_LAG_GT_COLS = True
    case _:
        USE_LAG_GT_COLS = False
```

The above loads the data of each table from an export of a sqlite database where project data was stored

Before explore the data further it helps to visualize the system work flow.  COMM DETECT Model Process Flow

CHUNKS Table

The chunks table holds the transcript data in ~15-second chunks: Both sports broadcast content and commercial content. Look at ground_truth_label_**c** to tell the difference between a commercial chunk (1) and a sports broadcast chunk (0). This is a binary label 1 or Also notice that 15-second chunks contain partial setences, continued in the next chunk.

In [6]:
```
print("EXAMPLE SPORTS BROADCASTING and COMMERCIAL Chunks")
rows_9_to_12 = df_chunks.iloc[9:13]  # iloc[start:end] includes start and excludes
pd.set_option('display.max_colwidth', 300)
rows_9_to_12[['chunk_id','cur_chunk', 'cur_chunk_start', 'cur_chunk_end', 'ground_t
```

```
EXAMPLE SPORTS BROADCASTING and COMMERCIAL Chunks
```

Out[6]:

	chunk_id	cur_chunk	cur_chunk_start	cur_chunk_end	ground_truth_label_c
9	9	he wanted. And if you watch how Steph Curry moves with the ball, without the ball, it doesn't matter, there is a fluid nature to the skill set that he brings to the table, and also how he just has a patience, a poise, a quickness, that he never gets fed up,	00:02:30.000279	00:02:45.000114	0
10	10	and he was lights out.\r\n\r\nThis was a loss, though, against the Hawks in OT. Curry goes for 60 points, 38 field goal attempts, he made 10 threes, the second oldest player with a 60-point game. The late	00:02:45.000115	00:03:00.000015	0
11	11	Kobe Bryant did it at the age of 37.\r\n\r\nCountdown to tipoff continues. Coming up, Bridges from downtown. Meghan has that story as we get you ready for the Nets and Warriors here on YES.	00:03:00.000016	00:03:15.000388	0
12	12	\r\n\r\nWelcome to the home of blackjack, with the largest library of blackjack games online, Spanish 21, Live Dealin', Vacation Blackjack, and more. Download America's number one online	00:03:15.000389	00:03:30.000303	1

Because a chunk can have sentences related to both commercials and sportsbroadcasts, we also have a multi-class ground_truth_label_**num** with values 0 to 4: Sports Broadast(0), Sports Broadast To Commercial(1), Commercial(2), Commercial To Sports Broadcast(3), and

Commercial to Another Commerical(4). See examples of transitional chunks below. Note: Any chunk that has any portion of a commercial has ground_truth_label_**c** of value 1.

In [7]:
```
print("EXAMPLE TRANSITIONAL CHUNKS")
df_sb_to_c = df_chunks[df_chunks['chunk_id'].isin([20,35,521,779])]
df_sb_to_c[['chunk_id','cur_chunk', 'cur_chunk_start', 'cur_chunk_end', 'ground_tru
```

```
EXAMPLE TRANSITIONAL CHUNKS
```

Out[7]:

	chunk_id	cur_chunk	cur_chunk_start	cur_chunk_end	ground_truth
20	20	celebrating 43 years in business. Let us renew your home.\r\n\r\nWelcome back to Barclays Center, as we get you set for a fun	00:05:15.000141	00:05:30.000365	
35	35	Injured? Visit forthepeople.com.\r\n\r\nWe're set to go, the Nets and the Warriors. Let's check out the starting lineups, presented by Webull, learning, sharing, investing.\r\n\r\nIt is	00:09:30.000187	00:09:45.000132	
521	521	Steph Curry end up coming up with it, get it ahead to Draymond Green, and it's Podziemski who's had a nice night here tonight at Brooklyn, 14 for him so far, and Warriors up 11.\r\n\r\nComing to Disney Plus...\r\n\r\nYour friend, Captain Marvel,	02:11:45.000256	02:12:00.000185	
779	779	Boston up by all of them.\r\n\r\nWhat do you do when your tire goes flat and there's no air anywhere to fix that? You reach for Bullseye Pro,	00:38:30.000063	00:38:45.000119	

One of our challenges was to convert these chunks to sentences.

TEXT_BLOCKS table added for the Cal Capstone project

The **TEXT_BLOCKS** table holds the sentences. We used the NLTK package to convert chunks to sentences. It required subtle code to link chunks to handle and complete partial

sentences. Each text block (sentence record) is linked to a parent chunk. **We ended up with 10,574 sentences to predict across 2,892 chunks**

In [8]:
```
print(df_chunks.shape[0])
print(df_text_blocks.shape[0])
```

```
2892
10754
```

While the code to convert chunks to sentences uses a database which makes it challenging to share in a re-usable notebook, here are code snippets. You can find the full code base {***HERE***}, but the installation and running the stage 1 large language model interaction is not feasible given authentication tokens and database installation required.

In [ ]:
```
```

The **text_blocks** table looks very close to the **chunks** table except it's key is block_id and it has a parent foreign key to the chunks table

So this is the text chunk from above for CHUNK ID 20:

> celebrating 43 years in business. Let us renew your home.\r\n\r\nWelcome back to Barclays Center, as we get you set for a fun

It gets transformed into text_blocks ...

In [9]:
```
df_same_chunks_as_sentences = df_text_blocks[df_text_blocks['chunk_id'].isin([20])]
print("CHUNK ID: 20 from above now represented as sentences in text_blocks table")
df_same_chunks_as_sentences.head()
```

```
CHUNK ID: 20 from above now represented as sentences in text_blocks table
```

Out[9]:

	block_id	session_id	sequence_num	trans_dt	source_id	cur_chunk	cur_chunk_
59	59	0	60	2024-04-27 20:50:54.685231	0	Franzoso Contracting, celebrating 43 years in business.	00:05:15.00
60	60	0	61	2024-04-27 20:50:54.698591	0	Let us renew your home.	00:05:15.00

Note that part of the chunk's text that was an incomplete sentence moved to the next chunk (21)

In [10]:
```
df_same_chunks_as_sentences = df_text_blocks[df_text_blocks['chunk_id'].isin([21])]
print("The last text fragment of CHUNK ID 20 is the beginning of the now complete f
```

```
df_same_chunks_as_sentences.head(1)
```

```
The last text fragment of CHUNK ID 20 is the beginning of the now complete first sen
tence in text_blocks table for CHUNK ID 21
```

Out[10]:

	block_id	session_id	sequence_num	trans_dt	source_id	cur_chunk	cur_chunk_s
61	61	0	62	2024-04-27 20:50:54.752588	0	Welcome back to Barclays Center, as we get you set for a fun Monday night matchup between the Brooklyn Nets and the Golden State Warriors.	00:05:30.000

GROUND TRUTH LABELING - TRANSITIONS TABLE

The next challenge is to set the 'ground truth' labels all of these chunks and sentences. This was done by manually watching the videos and manually entering the timestamp of transitions from one state to another. The states entered were: Sports Broadast, Sports Broadast To Commercial, Commercial, Commercial To Sports Broadcast, and Commercial to Another Commerical. They were translated to numbers 0 to 4, respectively.

The **transitions** table holds this information

In [11]:
```
df_transitions.head(3)
```

Out[11]:

	transition_id	transition_code	timestamp	transition_excel_filename	
0	0	COMMERCIAL-TO-ANOTHER COMMERCIAL	00:00:17.000000	transition_superbowl_2024.xlsx	C:\U
1	1	COMMERCIAL-TO-SPORTS_BROADCASTING	00:00:48.000000	transition_superbowl_2024.xlsx	C:\U
2	2	SPORTS_BROADCASTING-TO-COMMERCIAL	00:04:31.000000	transition_superbowl_2024.xlsx	C:\U

This approach allowed my python program to read the chunks table sequentially and using the transitions table, mark each record with one of the five values. Similar approach for the text_blocks table but because it is only at the sentence level, it stored the ground_truth_label_c (1 for commercial, 0 for sports broadcast)

LLM_PREDICTIONS TABLE

The most central table in the project is the **llm_predictions** table. It holds the the prediction output from chatgpt API about whether the text is a commercial or sports broadcast.

It also holds the statistics about the text block that we use to try to improve the chatgpt prediction using other ML Modeling techniques. The most important columns are:

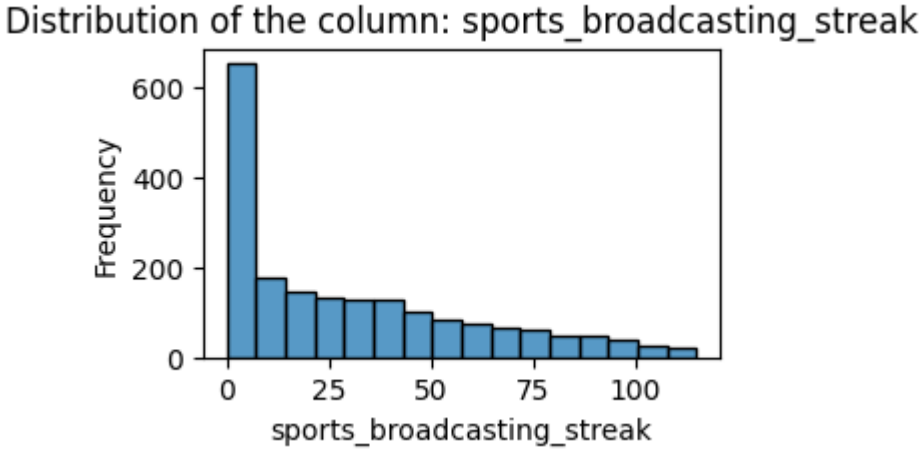
- response_classification_c: Stage 1 prediction - values 0 for 'Sports Broadcast' and 1 for 'Commercial'
- num_blocks_since_sb - Number of sentences since seen a sports broadcast
- num_blocks_since_c - Number of block since seen a commercial
- sb_blocks_so_far - total number of sport broadcast sentences seen so far in the game
- c_blocks_so_far - total number of commercial sentences seen so far in the game
- sports_broadcasting_streak - number of sports broadcast sentences seen in a row (including the given sentence of this row)
- commercial_streak - number of commercials sentences seen in a row (including the given sentence of this row)

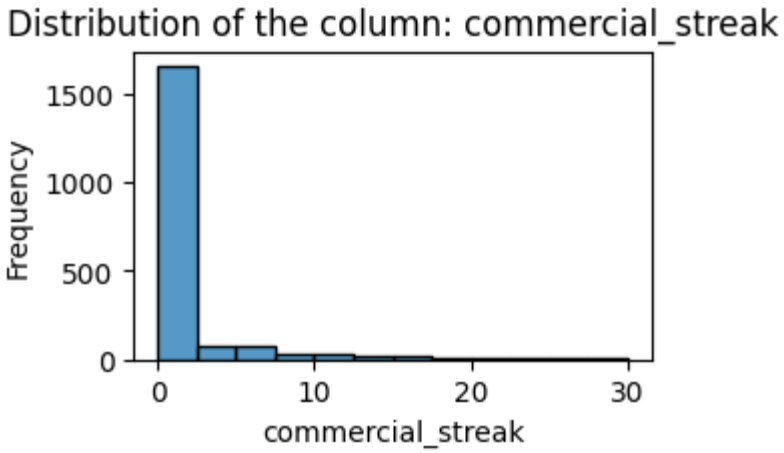
To explore this data, it can help to look at it per game. The games are delineated in the llm_predictions table by session_id.

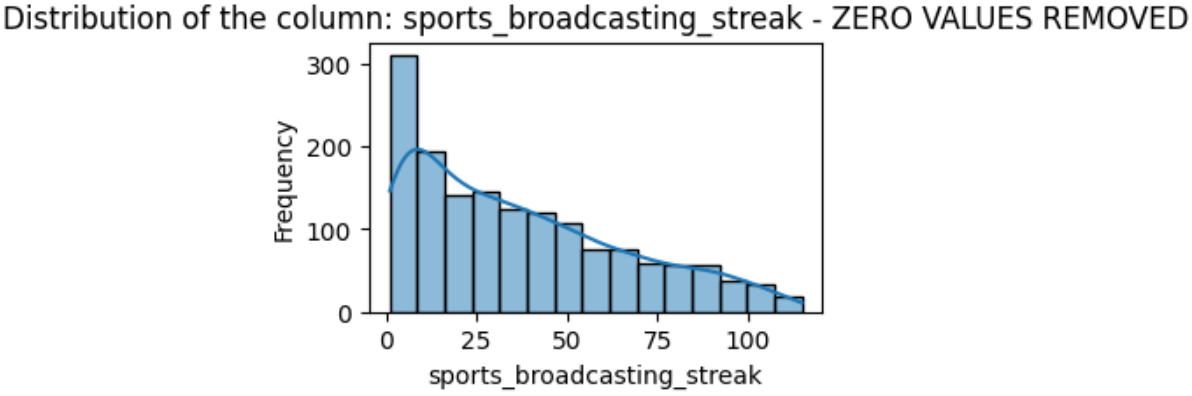
In [12]:
```
def my_plot_dist_df_cols(df, drop_zeros = False):
    # Create a distribution plot for column 'A'

    for col in df.columns:
        if drop_zeros:
            chart_df = df[df[col] != 0]
            zero_drop_str = " - ZERO VALUES REMOVED"
            show_kde = True
        else:
            chart_df = df
            zero_drop_str = ""
            show_kde = False
        plt.figure(figsize=(FIGX/3, FIGY/3))
        sns.histplot(chart_df[col], kde=show_kde)
        plt.title('Distribution of the column: ' +col +zero_drop_str)
        plt.xlabel(col)
        plt.ylabel('Frequency')
        plt.show()
```

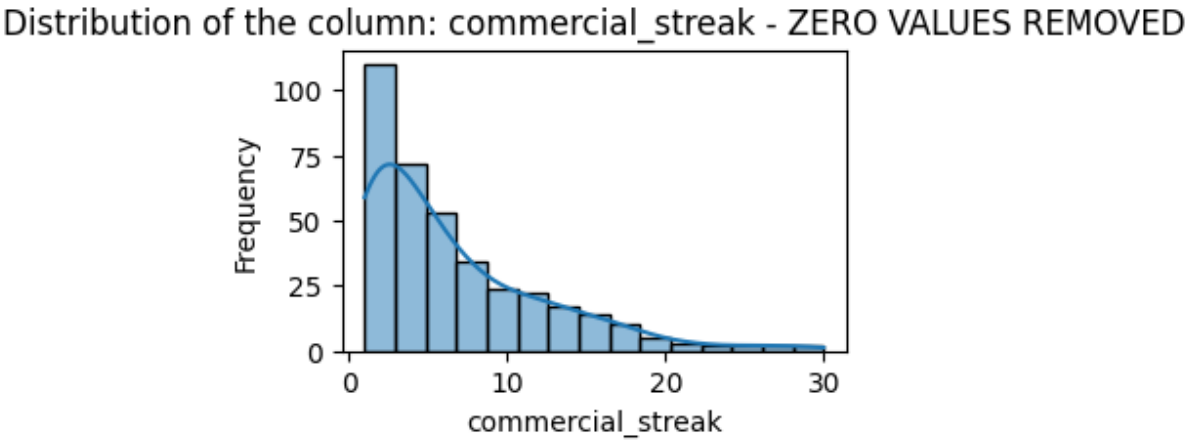
```python
def describe_game(df_llm_predictions, df_text_blocks, title_session, title):
    def_pred_tb = pd.merge(df_llm_predictions, df_text_blocks, left_on='cur_block_i
    print(title)
    df_game = def_pred_tb[def_pred_tb['session_id']==title_session]
    df_game_impt_flds = df_game[[ 'sports_broadcasting_streak', 'commercial_streak'
    my_plot_dist_df_cols(df_game_impt_flds, drop_zeros = False)
    my_plot_dist_df_cols(df_game_impt_flds, drop_zeros = True)
    #mins = df_game_important_fields.max(numeric_only=True)
    return
    #    return mins[['response_classification_num', 'response_classification_c', '

describe_game(df_llm_predictions, df_text_blocks, 0, "Stage 1 Predictions for Warri
```

```
Stage 1 Predictions for Warriors Vs Nets Basketball Game
```







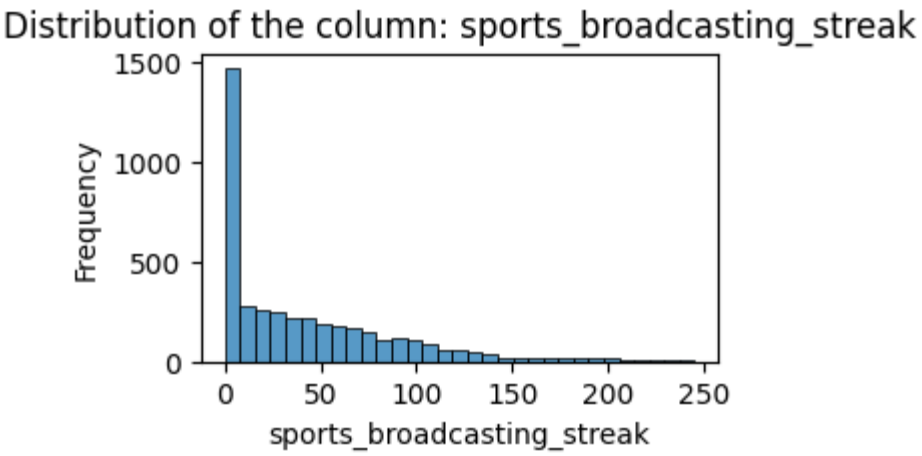


We hope these statistics and context about each sentence gives us features to help stage 2 model improve the initial prediction accuracy over what chatgpt predicted.

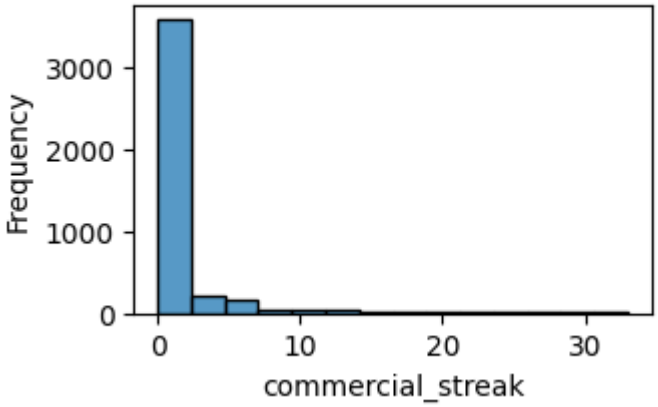
For example, will looking at commercial streaks (contigous sentences of commercials) for this game help us predict the current block? Here, we know that at some point there were a max 30 commercial sentences in a row and 115 sports broadcasting sentences in a row. Let's compare to another set of sentences of the total 4 sporting events in the data set.

In [13]:
```
describe_game(df_llm_predictions, df_text_blocks, 7, "Stage 1 Predictions for the S
```

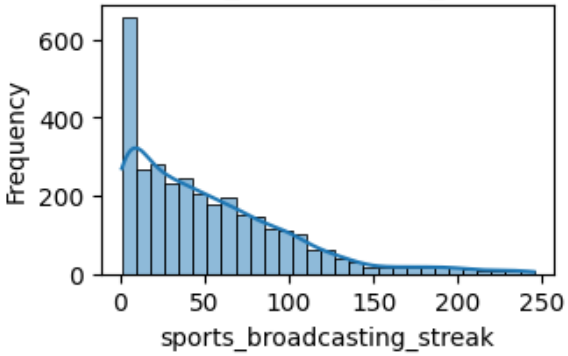
```
Stage 1 Predictions for the Super Bowl Football Game
```



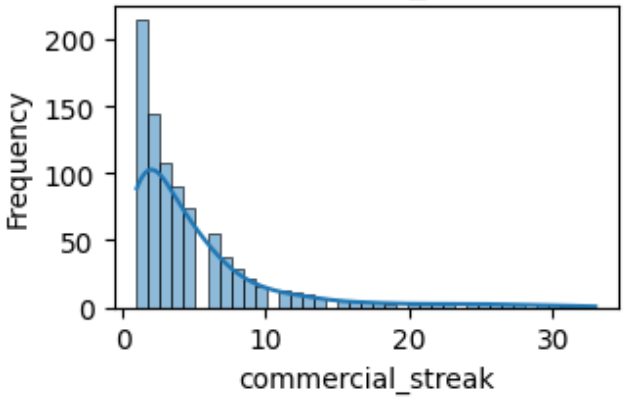
Distribution of the column: commercial_streak



Distribution of the column: sports_broadcasting_streak - ZERO VALUES REMOVED



Distribution of the column: commercial_streak - ZERO VALUES REMOVED



Before we go onto data exploration, let us, for full context, see the schema of all the data we are working with.

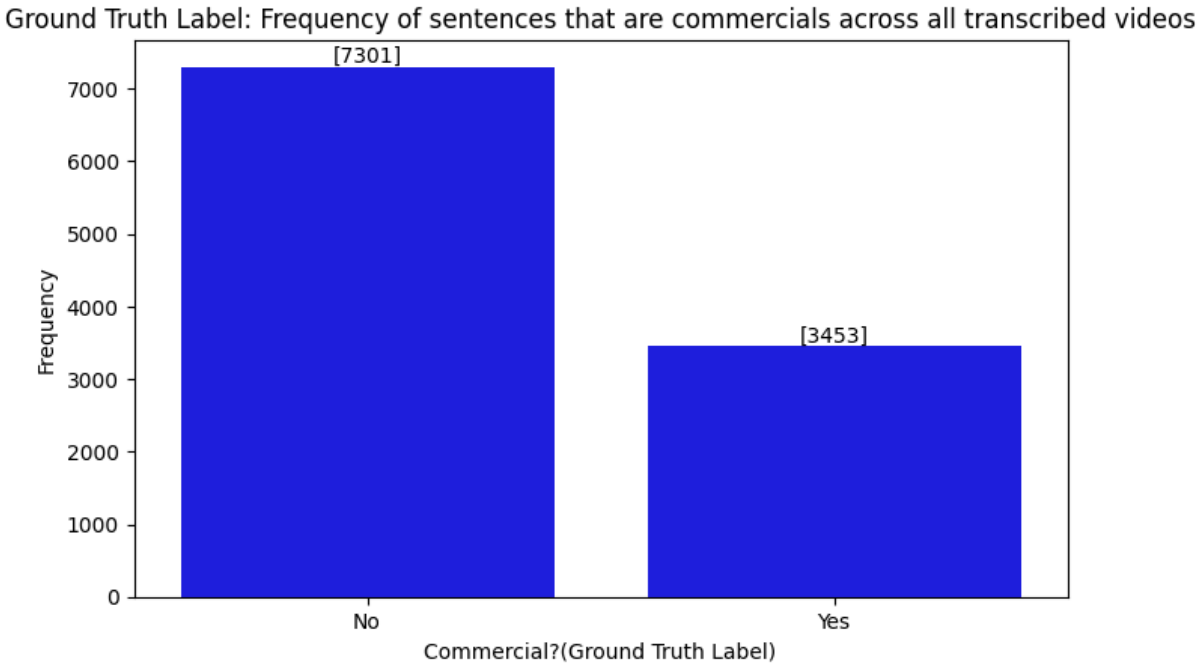

DB Schema

Data Exploration

Let's start by looking at the actual number of sentences that are commercials in our full data set.

In [14]:
```
def get_joined_df(df_llm_predictions, df_text_blocks, df_text_block_stat, session_i
    df_pred_tb = pd.merge(df_llm_predictions, df_text_blocks, left_on='cur_block_id
    df_pred_tb = pd.merge(df_pred_tb, df_text_block_stat, left_on='cur_block_id_use
    if session_id is not None:
        df_pred_tb_filtered = df_pred_tb[df_pred_tb['session_id'] == session_id]
    else:
        df_pred_tb_filtered = df_pred_tb
    df_len = len(df_pred_tb_filtered)
    df_start = int(round(df_len*start_at_pct,0))
    df_end = int(round(df_len*end_at_pct,0))
    # comment out if don't need to see column list
    # print(df_pred_tb_filtered.columns)
    return df_pred_tb_filtered.iloc[df_start:df_end]
```

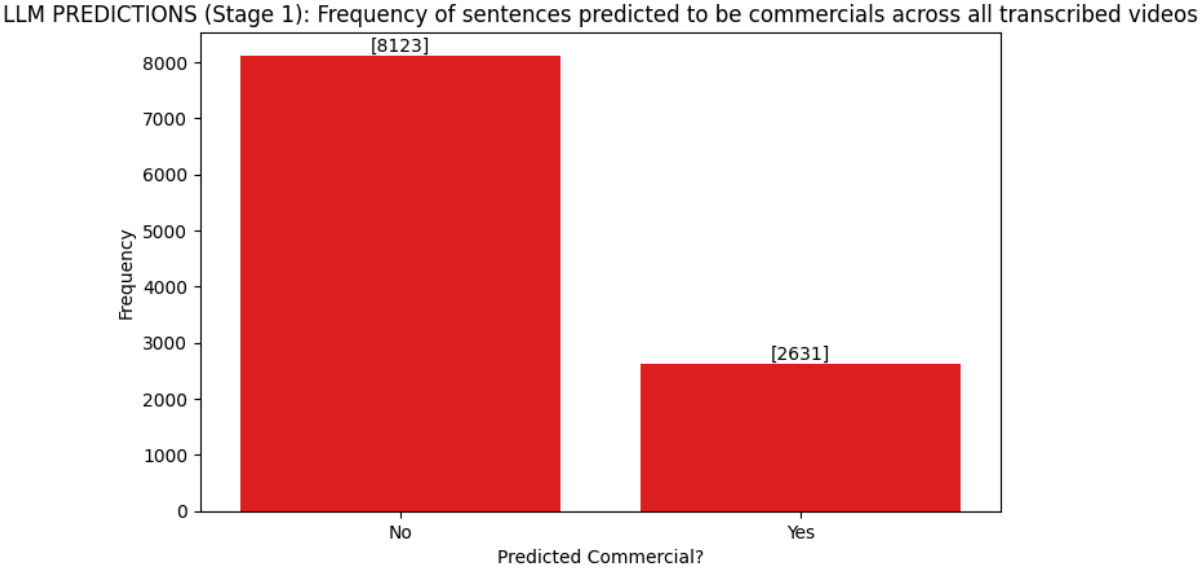
In [15]:
```
def my_hist_plot(edges, count, xlabel = 'Values', ylabel = 'Frequency', title_str =
# Create a DataFrame for the bar plot
    if bins_override_list is None:
        df = pd.DataFrame({
            'bin_center': (edges[:-1] + edges[1:]) / 2,
            'count': counts
            })
    else:
        df = pd.DataFrame({
            'bin_center': bins_override_list,
            'count': counts
            })
    # Initialize the plot
    plt.figure(figsize=(FIGX/1.25, FIGY/1.25))

    # Create the bar plot using sns.barplot()
    sns.barplot(data=df, x='bin_center', y='count', color=color)

    # Add the values at the top of the bars with a small offset
    for index, row in df[['count']].iterrows():
        plt.text(row.name, row.values + 0.5, row.values, color='black', ha="center"

    # Add labels and title
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.title(title_str)

# show ground truth labels
df_pred_tb = get_joined_df(df_llm_predictions, df_text_blocks, df_text_block_stat,
values = df_pred_tb['ground_truth_label_c']
bins = np.histogram_bin_edges(values, bins=2)
counts, edges = np.histogram(values, bins=bins)
# for this case, choose explicit values for X-axis
zero_and_one = ['No','Yes']
my_hist_plot(edges, counts, xlabel='Commercial?(Ground Truth Label)', title_str = '
```



As stated before, our project INPUT starts using the outputs of the LLM (ChatGPT) from Stage 1. It is our baseline from which we want to improve.

Let's see how frequently STAGE 1 predicts commercials.

In [16]:
```
# now show predicted values by stage 1
values = df_pred_tb['response_classification_c']
bins = np.histogram_bin_edges(values, bins=2)
counts, edges = np.histogram(values, bins=bins)
my_hist_plot(edges, counts, xlabel='Predicted Commercial?', title_str = 'LLM PREDIC
```



So we see that the new Stage 1 predicts 802 (3453-2631) fewer commercial sentences than actually are in the 722 minutes of

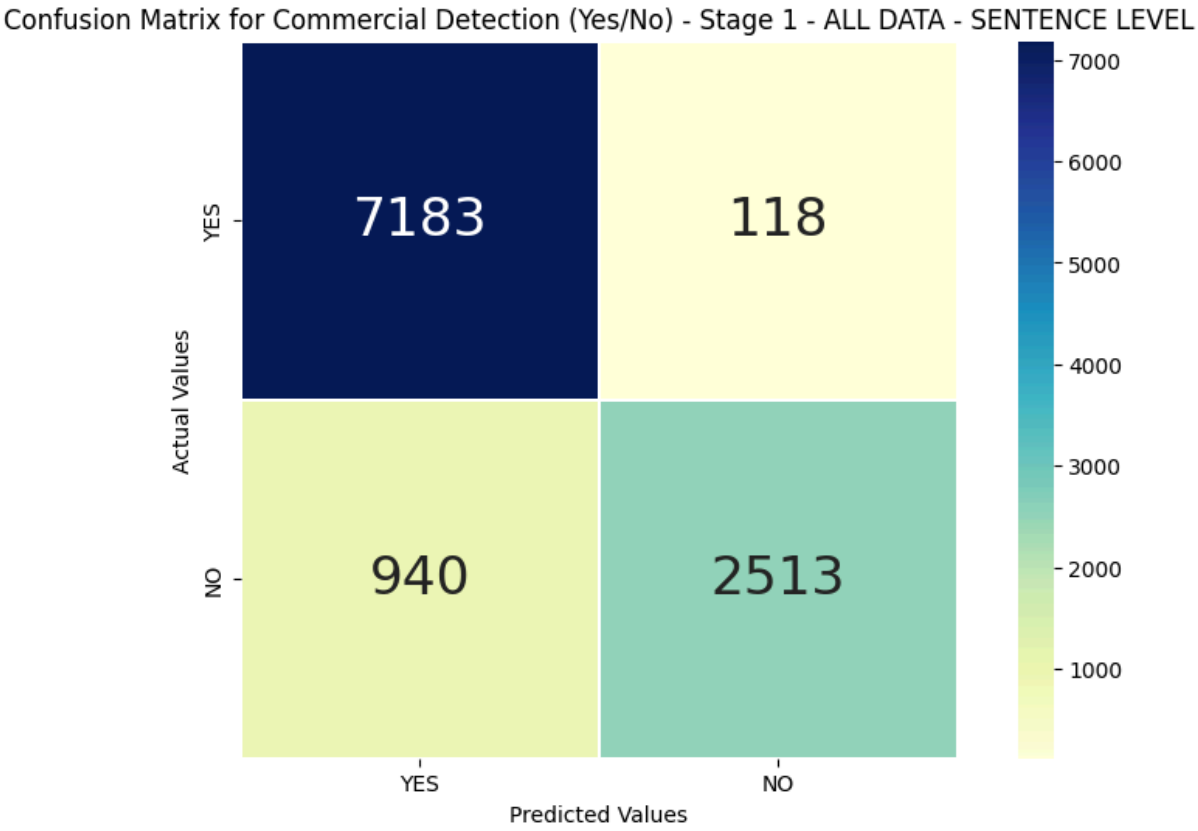
video.

Another baseline stat is the confusion matrix from stage 1

The new Stage 1 built for the Captsone project, besides working on a sentence level, also asks chatGPT API 3 times using 3 different prompts and then uses a weighted voting approach to choose the final response. Two of the prompts use only the current single sentence to predict. One of the prompts uses the current sentence plus the previous two sentences as context to predict. For more detail, see HERE.

In [17]:
```
def eval_accuracy( exp_name, stage, actual, predicted):
    cm = confusion_matrix(actual, predicted)

    # Check that resulting shape makes sense
    if (np.shape(cm) != (2,2)):
        print(f"unexpected shape of binary confusion matrix: {np.shape(cm)}. Skippi
        raise Exception('Unable to calculate CM')
    cm_df = pd.DataFrame(cm, index = ['YES','NO'],columns = ['YES','NO'])
    cm_title =  'Confusion Matrix for Commercial Detection (Yes/No)' + ' - ' + stag

    #Plotting the confusion matrix
    plt.figure(figsize=(FIGX,FIGY))
    sns.heatmap(cm_df,
        vmin=cm_df.values.min(),
        vmax=cm_df.values.max(),
        square=True,
        cmap="YlGnBu",
        linewidths=0.1,
        annot=True,
        fmt=f'.{0}f',
        annot_kws={"size": 35 / np.sqrt(len(cm_df))})

    plt.title(cm_title)
    plt.ylabel('Actual Values')
    plt.xlabel('Predicted Values')

    fig_file = f"./output/conf_mat_{stage}_{exp_name}.png"
    plt.savefig(fig_file)
    plt.show()
    plt.clf()

    recall_score_macro = recall_score(actual, predicted, average='macro')
    print(f"recall_score_macro_{stage} : {round(recall_score_macro,4)}" )

    precision_score_macro = precision_score(actual, predicted, average='macro')
    print(f"precision_score_macro_{stage} : {round(precision_score_macro,4)}" )

    f1_score_macro = f1_score(actual, predicted, average='macro')
    print(f"f1_score_macro_{stage} : {round(f1_score_macro,4)}" )

    bal_acc_score = balanced_accuracy_score( actual, predicted)
    print(f"bal_acc_score_{stage} : {round(bal_acc_score,4)}" )
```

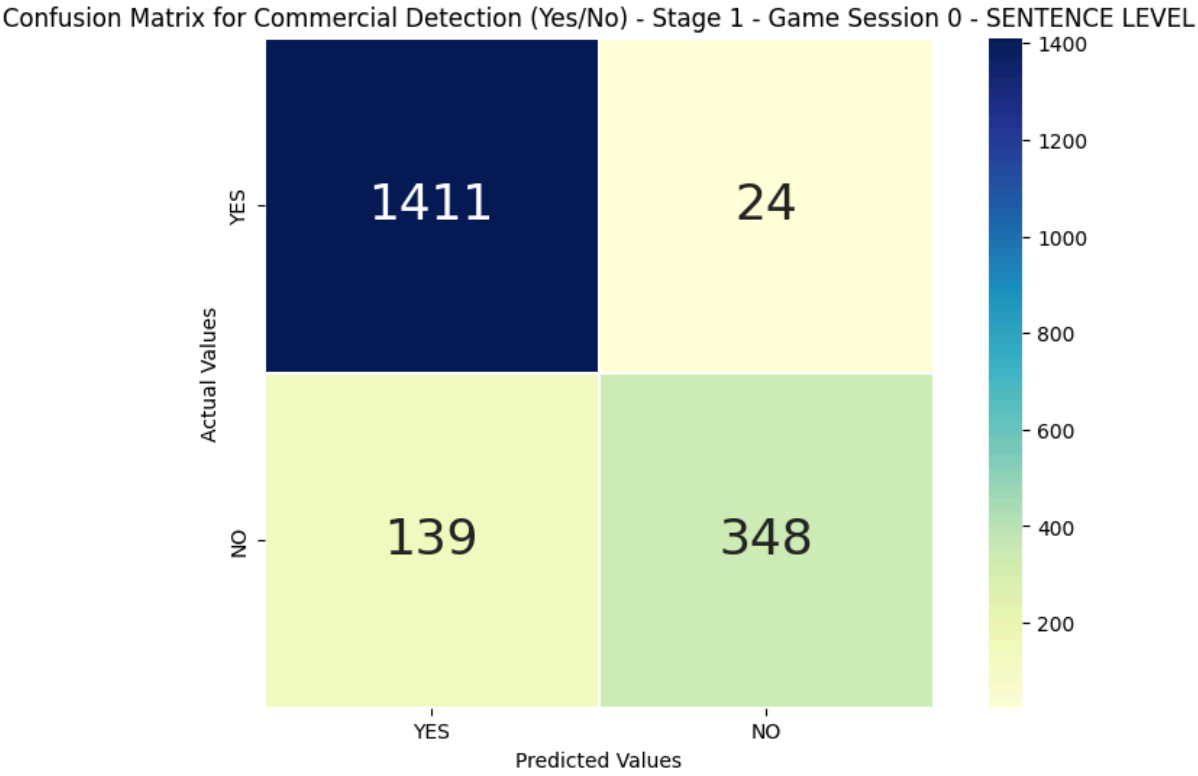
```
actual_binary = df_pred_tb['ground_truth_label_c']
pred_binary = df_pred_tb['response_classification_c']
eval_accuracy("All Experiments", "Stage 1 - ALL DATA - SENTENCE LEVEL", actual_bina
```




```
recall_score_macro_Stage 1 - ALL DATA - SENTENCE LEVEL : 0.8558
precision_score_macro_Stage 1 - ALL DATA - SENTENCE LEVEL : 0.9197
f1_score_macro_Stage 1 - ALL DATA - SENTENCE LEVEL : 0.8788
bal_acc_score_Stage 1 - ALL DATA - SENTENCE LEVEL : 0.8558
<Figure size 640x480 with 0 Axes>
```

METRICS FOR THE TEST DATA SET (BELOW) AND WILL USE THIS TO COMPARE TO STAGE 2 RESULTS

In [18]:
```
g0_actual_binary = df_pred_tb[df_pred_tb["session_id"] ==0]['ground_truth_label_c']
g0_pred_binary = df_pred_tb[df_pred_tb["session_id"] ==0]['response_classification_
eval_accuracy("All Experiments", "Stage 1 - Game Session 0 - SENTENCE LEVEL", g0_ac
```




```
recall_score_macro_Stage 1 - Game Session 0 - SENTENCE LEVEL : 0.8489
precision_score_macro_Stage 1 - Game Session 0 - SENTENCE LEVEL : 0.9229
f1_score_macro_Stage 1 - Game Session 0 - SENTENCE LEVEL : 0.8778
bal_acc_score_Stage 1 - Game Session 0 - SENTENCE LEVEL : 0.8489
<Figure size 640x480 with 0 Axes>
```

Let's explore the most interesting feature fields we can use for stage 2.

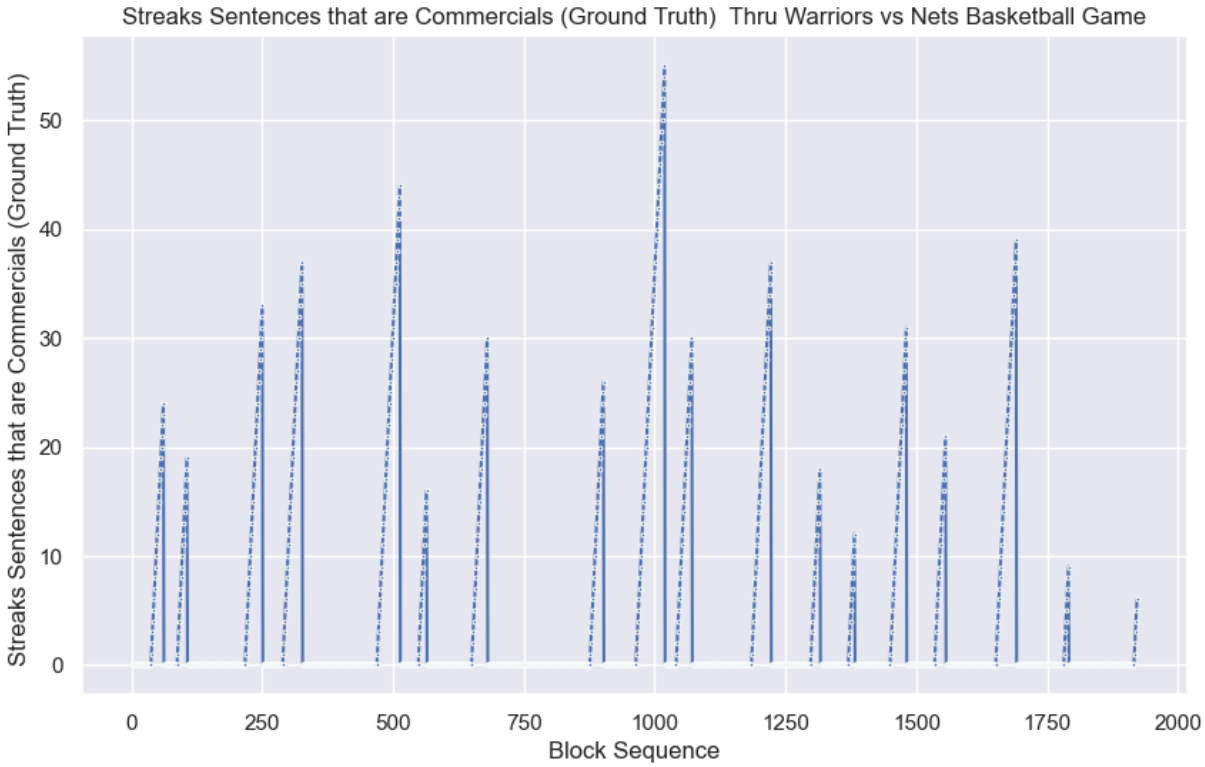
First, let's look at the commercial_streak field: My theory is that commercials run in bunches of contiguous groups and thus a long string of sequential sentences that are all commercials. I think we can use the fact that the last X sentences were predicted to be commercials to inform the stage 2 model about whether the current text block is a commercial or not.

Look at this chart of the actual commercial streaks:

(Filter to a particular game of the four games for this chart to make sense)

In [19]:
```
def chart_gt_y_vs_sequence(df_llm_predictions, df_text_blocks, df_text_block_stat,
    df_pred_tb_filtered = get_joined_df(df_llm_predictions, df_text_blocks, df_text

    # Set the aesthetic style of the plots
    sns.set(style="darkgrid")
    plt.figure(figsize=(FIGX,FIGY))
    sns.lineplot(x='sequence_num', y=y_field_name, data=df_pred_tb_filtered, marker
    # Add title and labels
    title_str = f"{y_field_title} Thru {session_title}"
    plt.title(title_str)
    plt.xlabel('Block Sequence')
    plt.ylabel(y_field_title)
```

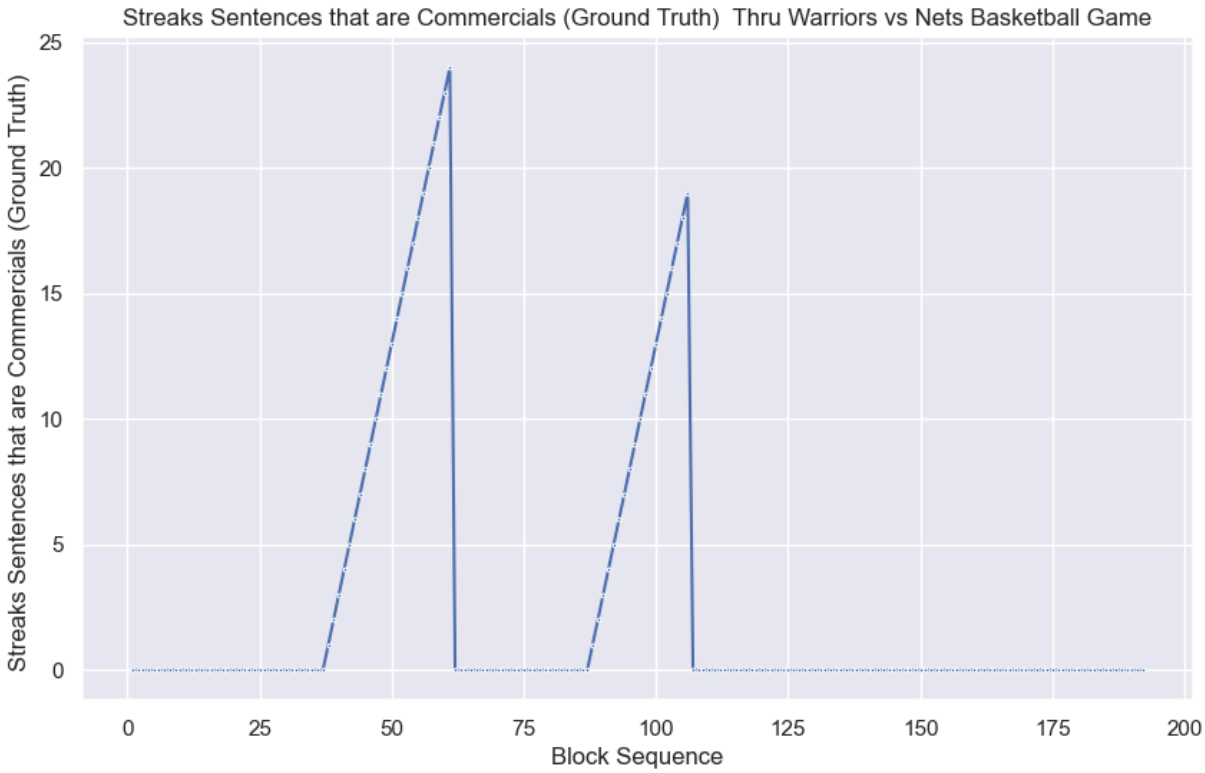
```
    # Show the plot
    plt.show()

chart_gt_y_vs_sequence(df_llm_predictions, df_text_blocks, df_text_block_stat, 0, '
```



In the above chart, see how the sequence or 'streak' of sequential commercial sentences grows and then starts again?

Visually, it looks like most commercials come in groups of 20 or more sentences. Let's zoom in a bit.

In [20]:
```
chart_gt_y_vs_sequence(df_llm_predictions, df_text_blocks, df_text_block_stat, 0, '
```

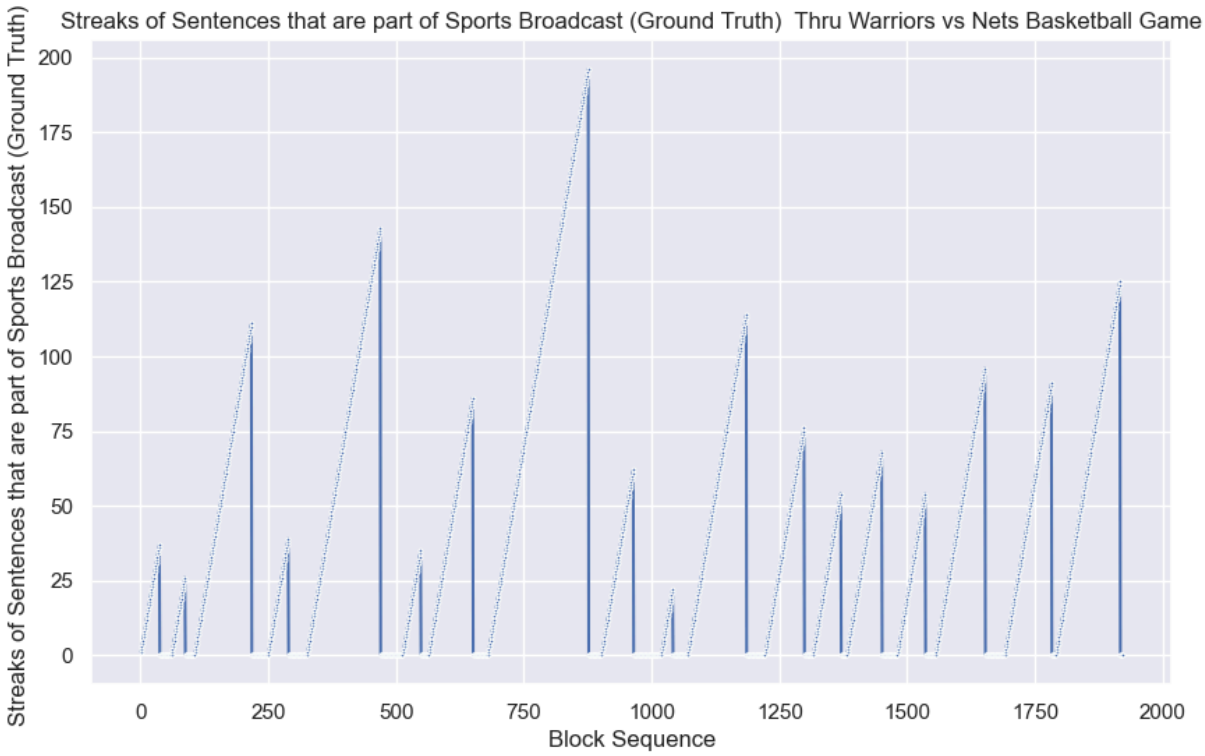


We see the same pattern with the sports broadcasting streaks in the gaps of the commercials.

The sports broadcast streak often reaches 50 sentences long or greater.

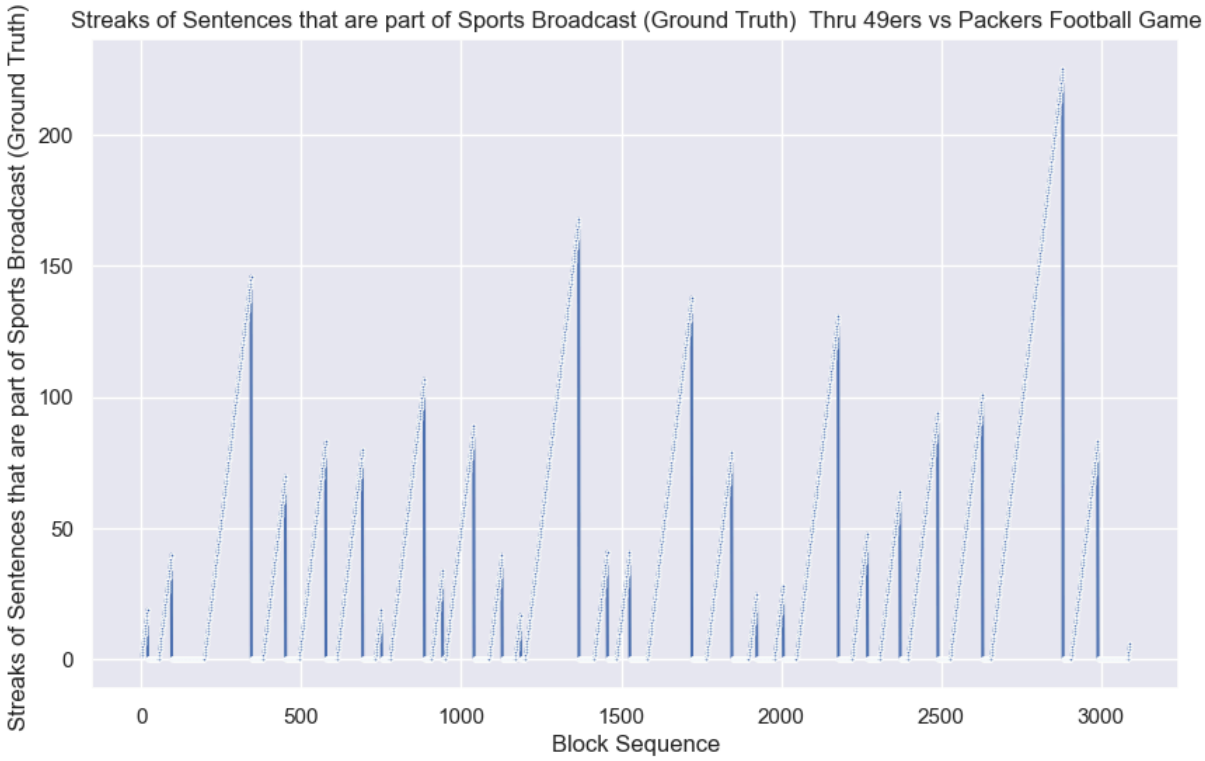
Both commercial streaks or sports broadcasting streaks might be useful signals to the stage 2 model.

In [21]:
```
chart_gt_y_vs_sequence(df_llm_predictions, df_text_blocks, df_text_block_stat, 0, '
```



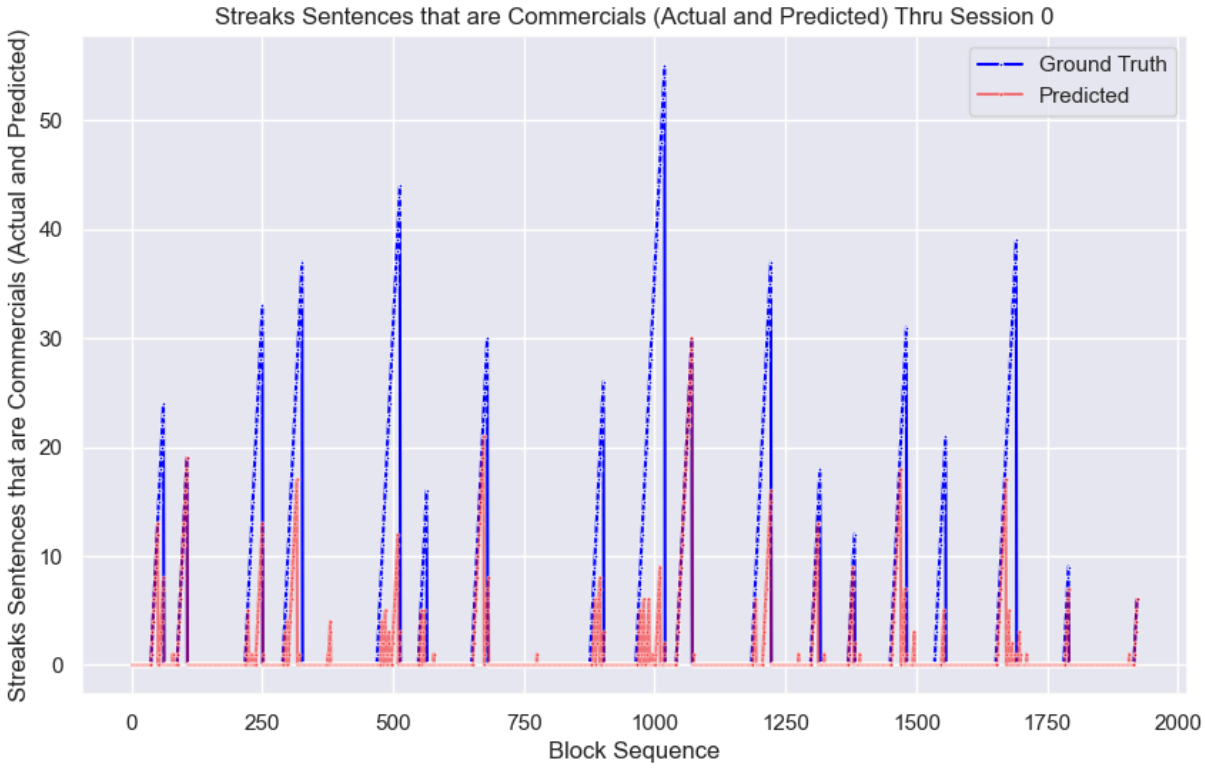
Same pattern for football games in the data set

```
In [22]: chart_gt_y_vs_sequence(df_llm_predictions, df_text_blocks, df_text_block_stat, 5, '
```



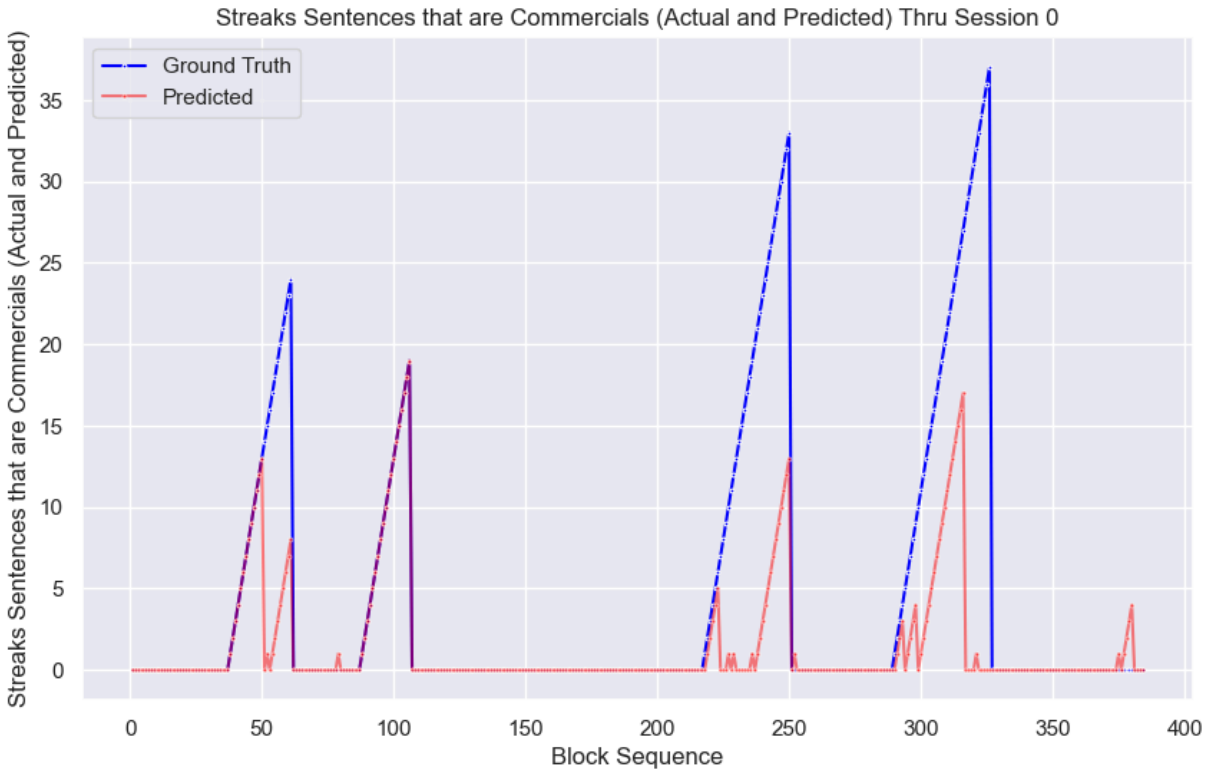
Yet, when we look at what Stage 1 predicted, the streaks are not consistently increasing. They rise and fall back 0 and start again multiple times inside the ground truth streaks triangles.

In [23]:
```
def chart_y_vs_sequence_p_and_gt(df_llm_predictions, df_text_blocks, df_text_block_
    df_pred_tb_filtered = get_joined_df(df_llm_predictions, df_text_blocks, df_text

    # Set the aesthetic style of the plots
    sns.set(style="darkgrid")
    plt.figure(figsize=(FIGX,FIGY))
    y_field_name_gt = y_field_name + '_gt'
    y_field_name_llm = y_field_name + '_llm'
    sns.lineplot(x='sequence_num', y=y_field_name_gt, data=df_pred_tb_filtered, mar
    sns.lineplot(x='sequence_num', y=y_field_name_llm, data=df_pred_tb_filtered, ma
    # Add title and labels
    title_str = f"{y_field_title} Thru {session_title}"
    plt.title(title_str)
    plt.xlabel('Block Sequence')
    plt.ylabel(y_field_title)

    # Show the plot
    plt.show()

chart_y_vs_sequence_p_and_gt(df_llm_predictions, df_text_blocks, df_text_block_stat
```



In [24]:
```
### Let's zoom in to see how the prediction flip-flops versus the actual (ground tr
```

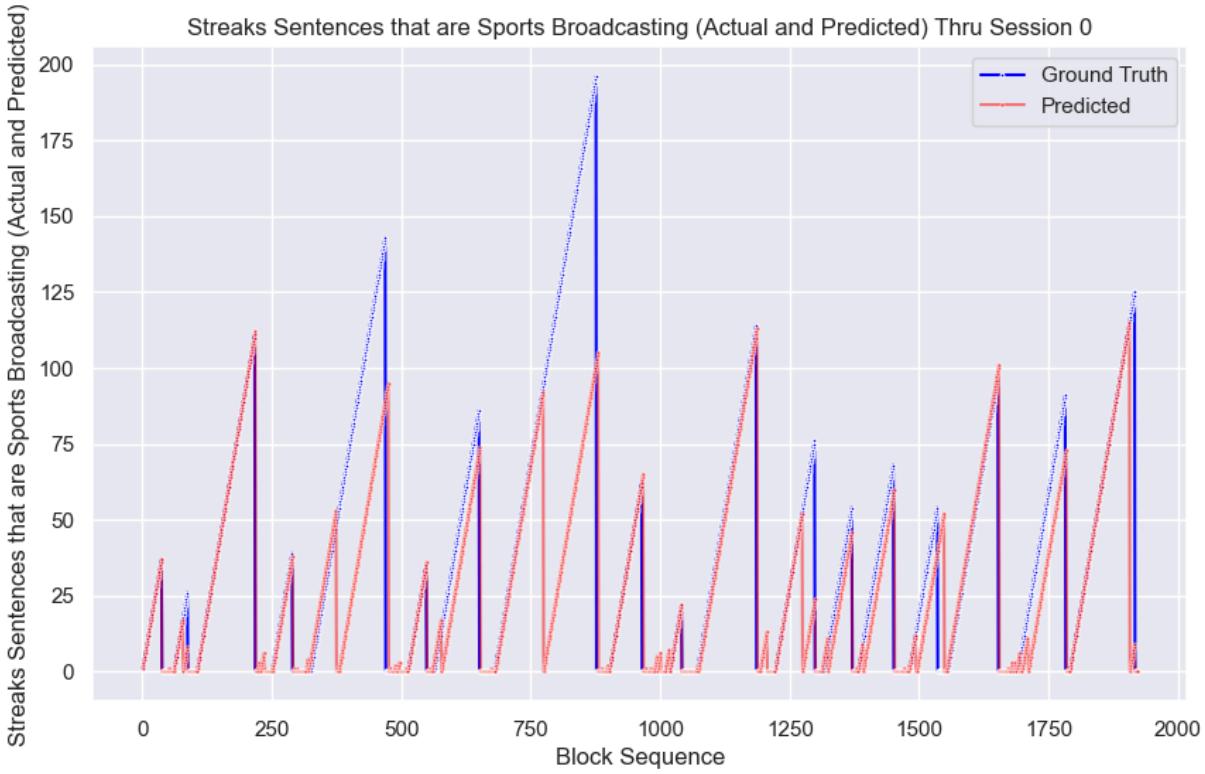
In [25]:
```
chart_y_vs_sequence_p_and_gt(df_llm_predictions, df_text_blocks, df_text_block_stat
```



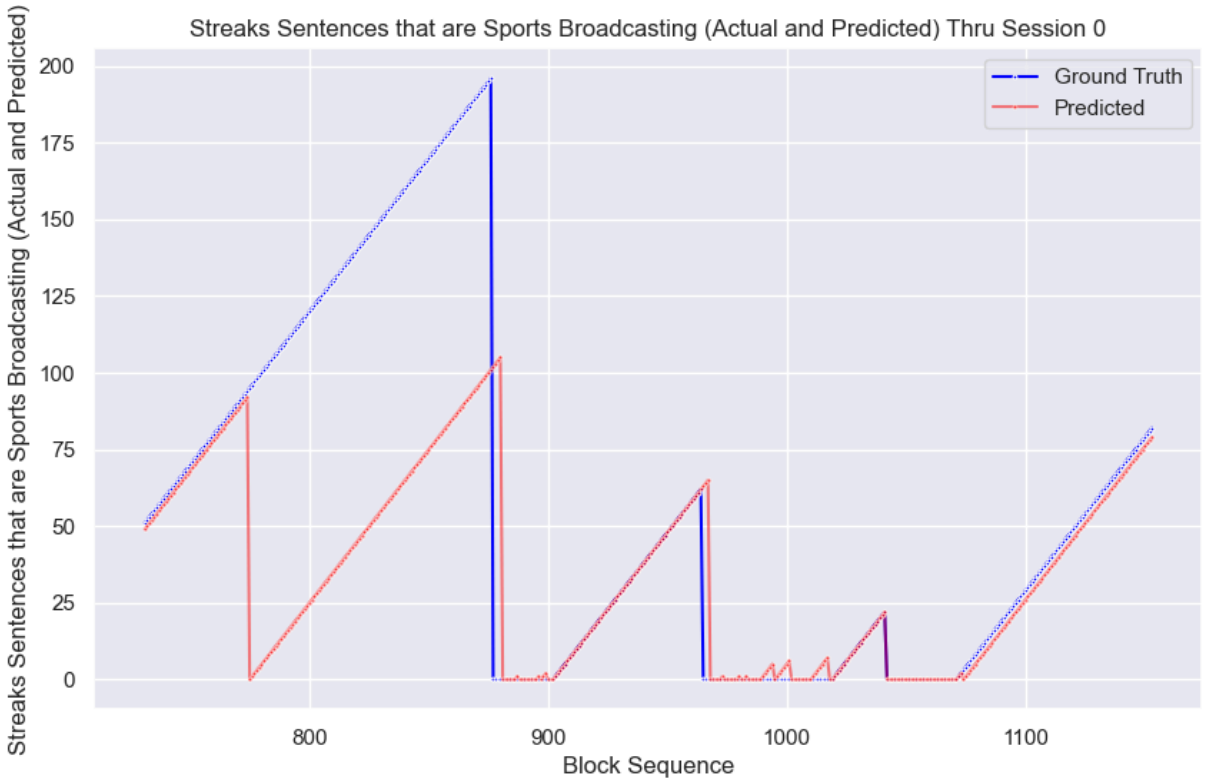
The same flip-flopping happens with sports broadcast streaks

You also see some errors outside the triangles

In [26]:
```
chart_y_vs_sequence_p_and_gt(df_llm_predictions, df_text_blocks, df_text_block_stat
```



In [27]:
```
chart_y_vs_sequence_p_and_gt(df_llm_predictions, df_text_blocks, df_text_block_stat
```



So we have a better understanding of the data and have explored some of its key characteristics.

Next is to work on feature engineering: generating potentially useful features from our existing ones and then trimming down to only the features that matter most

Feature Engineering

For a time-series based analysis it is often useful to create Lag, Diff, Rolling and Seasonal features. Will create several of each for the **target value llm_prediction.response_classification_c**

Note two assumptions to explore:

FIRST: To better simulate real-time conditions, we will first use **previous row's estimated category** rather than ground truth labels. In practice, this system might not have a real-time feedback loop to tell if the recent sentences are commercials or not.

SECOND: After that, we will see how well the system performs, if we could get the **previous sentence ground truth category** when trying to predict the current sentence.

Prepare dataframe for time series features

In [28]:
```
df_pred_tb_filtered = get_joined_df(df_llm_predictions, df_text_blocks, df_text_blo
```

In [29]:
```
df_pred_tb_filtered.shape[0]
```

Out[29]:
```
10754
```

Since we are analyzing a time-series data set we won't use random splits to avoid data leakage risks. We will split by session (or game).

We have 4 games in our data set and we will hold back one game (session_id=0) as the test set. The rest will be used for training. Session 0 holds 18% of the sentences.

In [30]:
```
df_pred_tb_filtered.groupby("session_id")["block_id_gt"].count()
```

Out[30]:
```
session_id
0    1922
2    1637
5    3089
7    4106
Name: block_id_gt, dtype: int64
```

In [31]:
```
def game_based_split(df_pred_filtered_tb_all_cols, game_session_id = TEST_SET_GAME_
    # lets cut down irrelevant or misleading columns
    # lets have a df that has all columns which can help with reporting later
    df_pred_filtered_tb_all_cols  = df_pred_tb_filtered.sort_values(by=["session_id

    print(f"test data to be based on game session: {game_session_id}")

    df_train_all_cols = df_pred_filtered_tb_all_cols[df_pred_filtered_tb_all_cols["
    print(f"Train rows = {df_train_all_cols.shape[0]}")

    df_test_all_cols = df_pred_filtered_tb_all_cols[df_pred_filtered_tb_all_cols["s
    print(f"Test rows = {df_test_all_cols.shape[0]}")

    return df_train_all_cols, df_test_all_cols

def get_df_base_cols(df_train_all_cols, df_test_all_cols, base_col_list = DEF_BASE_
    print("All columns listed below")
    print(df_train_all_cols.columns)
    print("---")
    print("Reducing to base feature columns but keeping parallel dataframe with all

    # remember to drop 'sesssion_id' from df feature list once we are done using it

    # but we will mainly use df with only relevant features (and later generated fe
    df_train_base_cols  = df_train_all_cols[base_col_list]
    df_test_base_cols  = df_test_all_cols[base_col_list]
    print(f"So starting with {len(df_train_base_cols.columns)-1} feature columns an
    print(df_train_base_cols.columns)
    df_test_base_cols.sample(1)
```

```
    return df_train_base_cols, df_test_base_cols
```

In [32]:
```
# Get all data we might need from join of data sets
df_pred_filtered_tb_all_cols  = df_pred_tb_filtered.sort_values(by=["session_id", "

# Remove one of the games to be our final test set
df_train_all_cols, df_test_all_cols = game_based_split(df_pred_filtered_tb_all_cols

# Narrow down columns to potential features and ground truth labels.
# Will narrow further but these two dataframes may be useful for reporting later.
if USE_LAG_GT_COLS:
    df_train_base_cols, df_test_base_cols = get_df_base_cols(df_train_all_cols, df_
else:
    df_train_base_cols, df_test_base_cols = get_df_base_cols(df_train_all_cols, df_
```

```
test data to be based on game session: 0
Train rows = 8832
Test rows = 1922
All columns listed below
Index(['Unnamed: 0', 'llm_predict_id', 'instr_prompt_used_name',
       'instr_prompt_used_text', 'cur_block_id_used', 'text_chunk_used',
       'llm_name', 'llm_other_params', 'fully_formed_request', 'requested_at',
       'response_is_valid', 'response_text', 'response_classification',
       'response_confidence', 'response_at', 'code', 'input_tokens',
       'output_tokens', 'sports_broadcasting_streak_llm',
       'commercial_streak_llm', 'sb_to_c_streak', 'c_to_c_streak',
       'c_to_sb_streak', 'c_blocks_so_far_llm', 'sb_blocks_so_far_llm',
       'num_blocks_since_sb_llm', 'num_blocks_since_c_llm', 'llm_exp_name',
       'response_classification_num', 'other_blocks_so_far', 'other_streak',
       'response_classification_c', 'llm_exec_time', 'windowed_resp_class_c',
       'lag_1_ground_truth_label_c', 'lag_2_ground_truth_label_c',
       'lag_3_ground_truth_label_c', 'lag_4_ground_truth_label_c',
       'lag_5_ground_truth_label_c', 'lag_10_ground_truth_label_c',
       'lag_15_ground_truth_label_c', 'lag_20_ground_truth_label_c',
       'sports_broadcasting_streak_lag_1_gt', 'num_blocks_since_sb_lag_1_gt',
       'commercial_streak_lag_1_gt', 'num_blocks_since_c_lag_1_gt',
       'sb_blocks_so_far_lag_1_gt', 'c_blocks_so_far_lag_1_gt', 'block_id_llm',
       'session_id', 'sequence_num', 'trans_dt', 'source_id', 'cur_chunk',
       'cur_chunk_start', 'cur_chunk_end', 'ground_truth_label',
       'ground_truth_label_num', 'ground_truth_label_c', 'chunk_id',
       'chunk_overlap_ind', 'text_block_stat_id',
       'sports_broadcasting_streak_gt', 'commercial_streak_gt',
       'c_blocks_so_far_gt', 'sb_blocks_so_far_gt', 'num_blocks_since_sb_gt',
       'num_blocks_since_c_gt', 'block_id_gt'],
      dtype='object')
---
Reducing to base feature columns but keeping parallel dataframe with all columns for
reporting later
So starting with 11 feature columns and one ground truth column
Index(['session_id', 'text_chunk_used', 'sports_broadcasting_streak_llm',
       'commercial_streak_llm', 'c_blocks_so_far_llm', 'sb_blocks_so_far_llm',
       'num_blocks_since_sb_llm', 'num_blocks_since_c_llm',
       'windowed_resp_class_c', 'sequence_num', 'response_classification_c',
       'ground_truth_label_c'],
      dtype='object')
```

NaN check BEFORE adding new features

In [33]:

```python
# count number of rows that have any NANs in any column
def count_nan_rows(df):
  """Counts the number of rows in a DataFrame that contain any NaN values."""
  return df.isna().any(axis=1).sum()

print("NaN check BEFORE FEATURE ENGINEERING")
train_nan_count = count_nan_rows(df_train_base_cols[DEF_BASE_COL_LIST_LLM])
print(f"Train set Nans:{train_nan_count}")
test_nan_count = count_nan_rows(df_test_base_cols[DEF_BASE_COL_LIST_LLM])
print(f"Train set Nans:{test_nan_count}")

train_tot_count = len(df_train_base_cols)
```

```
test_tot_count = len(df_test_base_cols)

train_nan_frac = train_nan_count/train_tot_count
print(f"% of training rows with at least one NaN: {train_nan_frac:.2f}")

test_nan_frac = test_nan_count/test_tot_count
print(f"% of test rows with at least one NaN: {test_nan_frac:.2f}")

if test_nan_frac <= NAN_TOLERANCE and train_nan_frac <= NAN_TOLERANCE:
    print("Leaving NaNs in the data sets because small number of them and XGBoost c
else:
    raise Exception("Too many NaNs")
```

```
NaN check BEFORE FEATURE ENGINEERING
Train set Nans:0
Train set Nans:0
% of training rows with at least one NaN: 0.00
% of test rows with at least one NaN: 0.00
Leaving NaNs in the data sets because small number of them and XGBoost can handle th
em
```

Create Rolling features

In [34]:

```
def create_rolling_col(df_train,df_test, col_to_roll, col_to_group_by, abs_window_v
    # just in case someone passes a negative lag value (protect from data leakage)
    abs_window_value = abs(abs_window_value)
    new_roll_col = f"{roll_col_prefix}_{abs_window_value}_{col_to_roll}"
    #print(new_roll_col)

    # ideally use passed function pointers but this works for now
    agg_func_name= roll_col_prefix
    df_new_train = df_train
    df_new_test = df_test
    if agg_func_name == "min":
        df_new_train[new_roll_col] = df_new_train.groupby(col_to_group_by)[col_to_r
        df_new_test[new_roll_col] = df_new_test.groupby(col_to_group_by)[col_to_rol
    elif agg_func_name == "max":
        df_new_train[new_roll_col] = df_new_train.groupby(col_to_group_by)[col_to_r
        df_new_test[new_roll_col] = df_new_test.groupby(col_to_group_by)[col_to_rol
    elif agg_func_name == "mean":
        df_new_train[new_roll_col] = df_new_train.groupby(col_to_group_by)[col_to_r
        df_new_test[new_roll_col] = df_new_test.groupby(col_to_group_by)[col_to_rol
    else:
        raise Exception("No known agg function for {agg_func_name}")

    return df_new_train,df_new_test

# Supported roll_col_prefix values: "min", "max", "mean"
def create_set_of_rolling_cols(df_train,df_test, col_to_roll, col_to_group_by, abs_
    df_train_w_lag_col = df_train
    df_test_w_lag_col = df_test
    for window_value in abs_window_value_list:
        df_train_w_roll_col, df_test_w_roll_col = create_rolling_col(df_train, df_t
                                                                     col_to_group_b
```

```
    return df_train_w_lag_col, df_test_w_lag_col

# Create rolling MINIMUM columns which we will "feature test" later for significanc
df_train_w_roll_cols1, df_test_w_roll_cols1 = create_set_of_rolling_cols(df_train_b
                                                "response_classification_c", "s

df_train_w_roll_cols, df_test_w_roll_cols = create_set_of_rolling_cols(df_train_w_r
                                                "windowed_resp_class_c", "sessi

# Create rolling MEAN columns which we will "feature test" later for significance
df_train_w_roll_cols2, df_test_w_roll_cols2 = create_set_of_rolling_cols(df_train_w
                                                "response_classification_c", "s

df_train_w_roll_cols, df_test_w_roll_cols = create_set_of_rolling_cols(df_train_w_r
                                                "windowed_resp_class_c", "sessi

df_train_w_roll_cols.sample(2)
```

Out[34]:

	session_id	text_chunk_used	sports_broadcasting_streak_llm	commercial_streak_llm	c
2679	5	This old, tired phone can't be traded in. That's a bit dramatic. I've stuck with it forever.	0	2	
4442	5	But the playoff success has eluded them in a big way and they've got to figure out why. Why it doesn't translate to the postseason? Aaron Jones back in the game.	151	0	

2 rows × 28 columns

Create Lag Statistic features

In [35]:

```
def create_lag_col(df_train,df_test, col_to_lag, col_to_group_by, abs_lag_value, la
    # just in case someone passes a negative lag value (protect from data leakage)
    abs_lag_value = abs(abs_lag_value)
    new_lag_col = f"{lag_col_prefix}_{abs_lag_value}_{col_to_lag}"
    #print(new_lag_col)

    df_new_train = df_train
    df_new_test = df_test
    df_new_train[new_lag_col] = df_new_train.groupby(col_to_group_by)[col_to_lag].s
    df_new_test[new_lag_col] = df_new_test.groupby(col_to_group_by)[col_to_lag].shi
```

```
    #return df_new_train.reset_index(),df_new_test.reset_index()
    return df_new_train,df_new_test

def create_set_of_lag_cols(df_train,df_test, col_to_lag, col_to_group_by, abs_lag_v
    df_train_w_lag_col = df_train
    df_test_w_lag_col = df_test
    for lag_value in abs_lag_value_list:
        df_train_w_lag_col, df_test_w_lag_col = create_lag_col(df_train, df_test, c

    return df_train_w_lag_col, df_test_w_lag_col

# Create lag columns which we will feature test later for significance
df_train_w_lag_cols1, df_test_w_lag_cols1 = create_set_of_lag_cols(df_train_w_roll_

df_train_w_lag_cols, df_test_w_lag_cols = create_set_of_lag_cols(df_train_w_lag_col

# Create lag columns on commercial streak and sports broadcasting streak so we see
df_train_w_lag_cols, df_test_w_lag_cols = create_set_of_lag_cols(df_train_w_roll_co
# note passing in the dfs received from previous create lag call into the next set
df_train_w_lag_cols, df_test_w_lag_cols = create_set_of_lag_cols(df_train_w_lag_col
```

In [36]:
```
print('now have generated features using roll and lag methods')
print(df_train_w_lag_cols.columns)
```

```
now have generated features using roll and lag methods
Index(['session_id', 'text_chunk_used', 'sports_broadcasting_streak_llm',
       'commercial_streak_llm', 'c_blocks_so_far_llm', 'sb_blocks_so_far_llm',
       'num_blocks_since_sb_llm', 'num_blocks_since_c_llm',
       'windowed_resp_class_c', 'sequence_num', 'response_classification_c',
       'ground_truth_label_c', 'min_3_response_classification_c',
       'min_10_response_classification_c', 'min_20_response_classification_c',
       'min_30_response_classification_c', 'min_3_windowed_resp_class_c',
       'min_10_windowed_resp_class_c', 'min_20_windowed_resp_class_c',
       'min_30_windowed_resp_class_c', 'mean_3_response_classification_c',
       'mean_10_response_classification_c',
       'mean_20_response_classification_c',
       'mean_30_response_classification_c', 'mean_3_windowed_resp_class_c',
       'mean_10_windowed_resp_class_c', 'mean_20_windowed_resp_class_c',
       'mean_30_windowed_resp_class_c', 'lag_1_response_classification_c',
       'lag_2_response_classification_c', 'lag_3_response_classification_c',
       'lag_4_response_classification_c', 'lag_5_response_classification_c',
       'lag_10_response_classification_c', 'lag_20_response_classification_c',
       'lag_30_response_classification_c', 'lag_1_windowed_resp_class_c',
       'lag_2_windowed_resp_class_c', 'lag_3_windowed_resp_class_c',
       'lag_4_windowed_resp_class_c', 'lag_5_windowed_resp_class_c',
       'lag_10_windowed_resp_class_c', 'lag_20_windowed_resp_class_c',
       'lag_30_windowed_resp_class_c', 'lag_1_commercial_streak_llm',
       'lag_2_commercial_streak_llm', 'lag_3_commercial_streak_llm',
       'lag_1_sports_broadcasting_streak_llm',
       'lag_2_sports_broadcasting_streak_llm',
       'lag_3_sports_broadcasting_streak_llm'],
      dtype='object')
```

Create some utility functions will need for FE

In [37]:

```
class LLMStatsUtils():
    def __init__(self, p_out_sb_streak_field = 'stage2_sports_broadcasting_streak',
                 p_out_num_blocks_since_sb_field = 'stage2_num_blocks_since_sb',
                 p_out_commercial_streak_field = 'stage2_commercial_streak',
                 p_out_num_blocks_since_c_field = 'stage2_num_blocks_since_c',
                 p_out_sb_blocks_so_far_field = 'stage2_sb_blocks_so_far',
                 p_out_c_blocks_so_far_field = 'stage2_c_blocks_so_far'):
        self.out_sb_streak_field = p_out_sb_streak_field
        self.out_num_blocks_since_sb_field = p_out_num_blocks_since_sb_field
        self.out_commercial_streak_field = p_out_commercial_streak_field
        self.out_num_blocks_since_c_field = p_out_num_blocks_since_c_field
        self.out_sb_blocks_so_far_field = p_out_sb_blocks_so_far_field
        self.out_c_blocks_so_far_field = p_out_c_blocks_so_far_field

    def re_init_stats(self):
        llm_stats = {'sports_broadcasting_streak':0,
                     'commercial_streak':0,
                     'c_blocks_so_far':0,
                     'sb_blocks_so_far':0,
                     'num_blocks_since_sb':0,
                     'num_blocks_since_c':0,
                     'llm_predict_id':-1,
                     'block_id':-1,
                     'text_block_stat_id':-1}


        return llm_stats

    def update_sb(self,llm_stats):
        llm_stats['sports_broadcasting_streak'] += 1
        llm_stats['num_blocks_since_sb'] = 0

        llm_stats['commercial_streak'] = 0

        llm_stats['num_blocks_since_c'] +=1

        llm_stats['sb_blocks_so_far'] +=1

        llm_stats['c_blocks_so_far'] = llm_stats['c_blocks_so_far']


    def update_c(self,llm_stats):
        llm_stats['sports_broadcasting_streak'] = 0
        llm_stats['num_blocks_since_sb'] += 1

        llm_stats['commercial_streak'] += 1

        llm_stats['num_blocks_since_c'] = 0

        llm_stats['sb_blocks_so_far'] = llm_stats['sb_blocks_so_far']

        llm_stats['c_blocks_so_far'] += 1
```

```
    def save_stats_to_df(self,llm_stats, index, df):
        #print(f"saving for index: {index}")
        df.at[index, self.out_sb_streak_field]  = llm_stats['sports_broadcasting_st
        df.at[index, self.out_num_blocks_since_sb_field] = llm_stats['num_blocks_si

        df.at[index, self.out_commercial_streak_field] = llm_stats['commercial_stre

        df.at[index, self.out_num_blocks_since_c_field] = llm_stats['num_blocks_sin

        df.at[index, self.out_sb_blocks_so_far_field] = llm_stats['sb_blocks_so_far

        df.at[index, self.out_c_blocks_so_far_field] = llm_stats['c_blocks_so_far']

        return df

    def update_llm_stats(self,in_df, stat_field = 'adj_resp_class_c'):
        out_df = in_df
        prev_session_id = -1
        out_df = out_df.sort_values(by=['session_id', 'sequence_num'])
        for index, row in out_df.iterrows():
            if row['session_id'] != prev_session_id:
                print('New session')
                llm_stats = self.re_init_stats()
                prev_session_id = row['session_id']
            else:
                if row['sequence_num'] != prev_sequence_num + 1:
                    raise Exception('Invalid order for llm prediction in update_llm

            if row[stat_field] == 0:
                self.update_sb(llm_stats)
            if row[stat_field] == 1:
                self.update_c(llm_stats)

            out_df = self.save_stats_to_df(llm_stats,index, out_df)
            prev_session_id = row['session_id']
            prev_sequence_num = row['sequence_num']

        return out_df
```

Create a several features based on hunches, visualization observations, and subject matter expertise

In [38]:

```
# Subject maatter expertise custom columns here.
def create_sme_cols(df_train_sme, df_test_sme, alpha = DEF_ALPHA):
    # TODO: If create any more SME columns, start wrapping into smaller functions s



    # create exponential decay feature on commercial_streak and sports_broadcasting

    # train
    df_train_sme["commercial_streak_decay"] = 1 - np.exp(-alpha*df_train_sme["comme
    df_train_sme["sports_broadcasting_streak_decay"] = 1 - np.exp(-alpha*df_train_s
    df_train_sme["lag_1_c_decay"] = 1 - np.exp(-alpha*df_train_sme["lag_1_commercia
    df_train_sme["lag_1_sb_decay"] = 1 - np.exp(-alpha*df_train_sme["lag_1_sports_b
   # test
```

```
    df_test_sme["commercial_streak_decay"] = 1 - np.exp(-alpha*df_test_sme["commerc
    df_test_sme["sports_broadcasting_streak_decay"] = 1 - np.exp(-alpha*df_test_sme
    df_test_sme["lag_1_c_decay"] = 1 - np.exp(-alpha*df_test_sme["lag_1_commercial_
    df_test_sme["lag_1_sb_decay"] = 1 - np.exp(-alpha*df_test_sme["lag_1_sports_bro

    # create characters per block feature
    # train
    df_train_sme["char_len"] = len(df_train_sme["text_chunk_used"])
    # test
    df_test_sme["char_len"] = len(df_test_sme["text_chunk_used"])

    # create square and square root of the _so_far and char_len features. Maybe it
    # train
    df_train_sme["c_blocks_so_far_squared"] = df_train_sme["c_blocks_so_far_llm"]**
    df_train_sme["sb_blocks_so_far_squared"] = df_train_sme["sb_blocks_so_far_llm"]
    df_train_sme["char_len_squared"] = df_train_sme["char_len"]**2
    df_train_sme["c_blocks_so_far_root"] = df_train_sme["c_blocks_so_far_llm"]**(1/
    df_train_sme["sb_blocks_so_far_root"] = df_train_sme["sb_blocks_so_far_llm"]**(
    df_train_sme["char_len_root"] = df_train_sme["char_len"]**(1/2)

    #test
    df_test_sme["c_blocks_so_far_squared"] = df_test_sme["c_blocks_so_far_llm"]**2
    df_test_sme["sb_blocks_so_far_squared"] = df_test_sme["sb_blocks_so_far_llm"]**
    df_test_sme["char_len_squared"] = df_test_sme["char_len"]**2
    df_test_sme["c_blocks_so_far_root"] = df_test_sme["c_blocks_so_far_llm"]**(1/2)
    df_test_sme["sb_blocks_so_far_root"] = df_test_sme["sb_blocks_so_far_llm"]**(1/
    df_test_sme["char_len_root"] = df_test_sme["char_len"]**(1/2)
    return df_train_sme, df_test_sme

# rename df for this section based on df_train_w_lag_calls
df_train_sme, df_test_sme = create_sme_cols(df_train_w_lag_cols, df_test_w_lag_cols

# see where we are
print(f"now have {len(df_train_sme.columns) - 1} feature columns")
print(df_train_sme.columns)
#df_train_sme[df_train_sme['c_streak_ended_quickly_lag_1'] == 1][['c_streak_ended_q
```

```
now have 60 feature columns
Index(['session_id', 'text_chunk_used', 'sports_broadcasting_streak_llm',
       'commercial_streak_llm', 'c_blocks_so_far_llm', 'sb_blocks_so_far_llm',
       'num_blocks_since_sb_llm', 'num_blocks_since_c_llm',
       'windowed_resp_class_c', 'sequence_num', 'response_classification_c',
       'ground_truth_label_c', 'min_3_response_classification_c',
       'min_10_response_classification_c', 'min_20_response_classification_c',
       'min_30_response_classification_c', 'min_3_windowed_resp_class_c',
       'min_10_windowed_resp_class_c', 'min_20_windowed_resp_class_c',
       'min_30_windowed_resp_class_c', 'mean_3_response_classification_c',
       'mean_10_response_classification_c',
       'mean_20_response_classification_c',
       'mean_30_response_classification_c', 'mean_3_windowed_resp_class_c',
       'mean_10_windowed_resp_class_c', 'mean_20_windowed_resp_class_c',
       'mean_30_windowed_resp_class_c', 'lag_1_response_classification_c',
       'lag_2_response_classification_c', 'lag_3_response_classification_c',
       'lag_4_response_classification_c', 'lag_5_response_classification_c',
       'lag_10_response_classification_c', 'lag_20_response_classification_c',
       'lag_30_response_classification_c', 'lag_1_windowed_resp_class_c',
       'lag_2_windowed_resp_class_c', 'lag_3_windowed_resp_class_c',
       'lag_4_windowed_resp_class_c', 'lag_5_windowed_resp_class_c',
       'lag_10_windowed_resp_class_c', 'lag_20_windowed_resp_class_c',
       'lag_30_windowed_resp_class_c', 'lag_1_commercial_streak_llm',
       'lag_2_commercial_streak_llm', 'lag_3_commercial_streak_llm',
       'lag_1_sports_broadcasting_streak_llm',
       'lag_2_sports_broadcasting_streak_llm',
       'lag_3_sports_broadcasting_streak_llm', 'commercial_streak_decay',
       'sports_broadcasting_streak_decay', 'lag_1_c_decay', 'lag_1_sb_decay',
       'char_len', 'c_blocks_so_far_squared', 'sb_blocks_so_far_squared',
       'char_len_squared', 'c_blocks_so_far_root', 'sb_blocks_so_far_root',
       'char_len_root'],
      dtype='object')
```

Feature Selection using Sequential Feature Selection SFS

First create several simple wrapper functions so can iterate and explore different features

In [39]:
```python
# Before creating X and y dataframes, you can trim down total features to be select

def final_trim_of_features(df_train, df_test, keep_these_cols):
    if keep_these_cols is None:
        return df_train, df_test
    else:
        return df_train[keep_these_cols], df_test[keep_these_cols]

def drop_helper_features(df_train_sme, df_test_sme):
    df_train_sme[['text_chunk_used','session_id']].head()
    # drop a few helper fields we don't need any more that would confuse the model
    #train
    df_train_after_fe = df_train_sme.drop(['text_chunk_used','session_id'], axis=1)
    #test
    df_test_after_fe = df_test_sme.drop(['text_chunk_used','session_id'], axis=1)
```

```
    return df_train_after_fe, df_test_after_fe
```

Now, trim columns you want to evaluate, select best ones to feed into the final model

In [40]:

```
def get_X_y(df_train_after_fe, df_test_after_fe):
    # create the separate Feature (X) and label(y) dataframes
    #train
    X_train = df_train_after_fe.drop('ground_truth_label_c', axis=1)
    y_train = df_train_after_fe['ground_truth_label_c']
    #test
    X_test = df_test_after_fe.drop('ground_truth_label_c', axis=1)
    y_test = df_test_after_fe['ground_truth_label_c']

    y_train.head()
    return X_train, y_train, X_test, y_test

def get_feature_selection(X_train, y_train):

    # feature count check
    if X_train.shape[1] <= DEF_NUM_FEATURES:
        num_features_to_select = X_train.shape[1] - 1
    else:
        num_features_to_select = DEF_NUM_FEATURES

    # scale
    scaler = StandardScaler()
    # Fit the scaler and transform the features
    X_train_scaled = scaler.fit_transform(X_train, y_train)

    # Create a time series cross validation splitter
    tscv = TimeSeriesSplit()

    # instantiate model we will use to search for best features
    model_instance = xgb.XGBClassifier()

    # Initialize the Sequential Feature Selector
    sfs = SequentialFeatureSelector(model_instance, n_features_to_select=num_featur
                                    scoring=FEAT_SCORING_METRIC, cv=tscv, n_jobs =

    # Fit the SFS using the training data
    sfs.fit(X_train_scaled, y_train)

    # Get the selected feature indices
    selected_indices = sfs.get_support(indices=True)

    print("Selected features indices:", selected_indices)
    # Map selected feature indices to column names
    selected_columns = X_train.columns[selected_indices]
    print(selected_columns)


    # Transform the training and test sets to contain only the selected features
    X_train_selected = sfs.transform(X_train)
```

```
    X_test_selected = sfs.transform(X_test)

    return selected_indices, selected_columns, X_train_selected, X_test_selected
```

In [41]:

```
# trim columns down for experiments in feature selection and model efficiency
print(df_train_sme.columns)

df_train_wo_helper, df_test_wo_helper = drop_helper_features(df_train_sme, df_test_

# MODIFY THIS LIST TO EXPIERMENT
# Not setting this variable (i.e., setting to None) means to keep ALL columns!

match RUN_MODE:
    case 'C':
        cols_to_keep = ['sports_broadcasting_streak_llm', 'commercial_streak_llm',
       'c_blocks_so_far_llm', 'sb_blocks_so_far_llm',
       'num_blocks_since_sb_llm', 'num_blocks_since_c_llm',
       'windowed_resp_class_c', 'sequence_num', 'response_classification_c',
       'ground_truth_label_c']
    case 'D':
        cols_to_keep = ['sports_broadcasting_streak_llm', 'commercial_streak_llm',
       'c_blocks_so_far_llm', 'sb_blocks_so_far_llm',
       'num_blocks_since_sb_llm', 'num_blocks_since_c_llm',
       'sequence_num', 'response_classification_c',
       'ground_truth_label_c', 'min_3_response_classification_c',
       'min_10_response_classification_c', 'min_20_response_classification_c',
       'min_30_response_classification_c',
       'mean_10_response_classification_c',
       'mean_20_response_classification_c',
       'mean_30_response_classification_c',
       'lag_1_response_classification_c',
       'lag_2_response_classification_c', 'lag_3_response_classification_c',
       'lag_4_response_classification_c', 'lag_5_response_classification_c',
       'lag_10_response_classification_c', 'lag_20_response_classification_c',
       'lag_30_response_classification_c',
       'lag_1_commercial_streak_llm',
       'lag_2_commercial_streak_llm', 'lag_3_commercial_streak_llm',
       'lag_1_sports_broadcasting_streak_llm',
       'lag_2_sports_broadcasting_streak_llm',
       'lag_3_sports_broadcasting_streak_llm', 'commercial_streak_decay',
       'sports_broadcasting_streak_decay', 'lag_1_c_decay', 'lag_1_sb_decay',
       'char_len', 'c_blocks_so_far_squared', 'sb_blocks_so_far_squared',
       'char_len_squared', 'c_blocks_so_far_root', 'sb_blocks_so_far_root',
       'char_len_root']
    # if pass None, then everything is kept
    case 'E':
        cols_to_keep = None
    case 'F':
        cols_to_keep = None
    case 'G':
        cols_to_keep = None
    case 'H':
        cols_to_keep = None
    case _:
        cols_to_keep = None
```

```
# if pass None, then everything is kept
df_train, df_test = final_trim_of_features(df_train_wo_helper,df_test_wo_helper, co


# create a copy so we can repeat modeling analysis for logistic regression later
df_train_lr = df_train.copy()
df_test_lr = df_test.copy()

print(df_train.columns)

# finally get X and y for the main model (XGBoost)
X_train, y_train, X_test, y_test = get_X_y(df_train, df_test)

print(X_train.columns)
```

```
Index(['session_id', 'text_chunk_used', 'sports_broadcasting_streak_llm',
       'commercial_streak_llm', 'c_blocks_so_far_llm', 'sb_blocks_so_far_llm',
       'num_blocks_since_sb_llm', 'num_blocks_since_c_llm',
       'windowed_resp_class_c', 'sequence_num', 'response_classification_c',
       'ground_truth_label_c', 'min_3_response_classification_c',
       'min_10_response_classification_c', 'min_20_response_classification_c',
       'min_30_response_classification_c', 'min_3_windowed_resp_class_c',
       'min_10_windowed_resp_class_c', 'min_20_windowed_resp_class_c',
       'min_30_windowed_resp_class_c', 'mean_3_response_classification_c',
       'mean_10_response_classification_c',
       'mean_20_response_classification_c',
       'mean_30_response_classification_c', 'mean_3_windowed_resp_class_c',
       'mean_10_windowed_resp_class_c', 'mean_20_windowed_resp_class_c',
       'mean_30_windowed_resp_class_c', 'lag_1_response_classification_c',
       'lag_2_response_classification_c', 'lag_3_response_classification_c',
       'lag_4_response_classification_c', 'lag_5_response_classification_c',
       'lag_10_response_classification_c', 'lag_20_response_classification_c',
       'lag_30_response_classification_c', 'lag_1_windowed_resp_class_c',
       'lag_2_windowed_resp_class_c', 'lag_3_windowed_resp_class_c',
       'lag_4_windowed_resp_class_c', 'lag_5_windowed_resp_class_c',
       'lag_10_windowed_resp_class_c', 'lag_20_windowed_resp_class_c',
       'lag_30_windowed_resp_class_c', 'lag_1_commercial_streak_llm',
       'lag_2_commercial_streak_llm', 'lag_3_commercial_streak_llm',
       'lag_1_sports_broadcasting_streak_llm',
       'lag_2_sports_broadcasting_streak_llm',
       'lag_3_sports_broadcasting_streak_llm', 'commercial_streak_decay',
       'sports_broadcasting_streak_decay', 'lag_1_c_decay', 'lag_1_sb_decay',
       'char_len', 'c_blocks_so_far_squared', 'sb_blocks_so_far_squared',
       'char_len_squared', 'c_blocks_so_far_root', 'sb_blocks_so_far_root',
       'char_len_root'],
      dtype='object')
Index(['sports_broadcasting_streak_llm', 'commercial_streak_llm',
       'c_blocks_so_far_llm', 'sb_blocks_so_far_llm',
       'num_blocks_since_sb_llm', 'num_blocks_since_c_llm',
       'windowed_resp_class_c', 'sequence_num', 'response_classification_c',
       'ground_truth_label_c', 'min_3_response_classification_c',
       'min_10_response_classification_c', 'min_20_response_classification_c',
       'min_30_response_classification_c', 'min_3_windowed_resp_class_c',
       'min_10_windowed_resp_class_c', 'min_20_windowed_resp_class_c',
       'min_30_windowed_resp_class_c', 'mean_3_response_classification_c',
       'mean_10_response_classification_c',
       'mean_20_response_classification_c',
       'mean_30_response_classification_c', 'mean_3_windowed_resp_class_c',
       'mean_10_windowed_resp_class_c', 'mean_20_windowed_resp_class_c',
       'mean_30_windowed_resp_class_c', 'lag_1_response_classification_c',
       'lag_2_response_classification_c', 'lag_3_response_classification_c',
       'lag_4_response_classification_c', 'lag_5_response_classification_c',
       'lag_10_response_classification_c', 'lag_20_response_classification_c',
       'lag_30_response_classification_c', 'lag_1_windowed_resp_class_c',
       'lag_2_windowed_resp_class_c', 'lag_3_windowed_resp_class_c',
       'lag_4_windowed_resp_class_c', 'lag_5_windowed_resp_class_c',
       'lag_10_windowed_resp_class_c', 'lag_20_windowed_resp_class_c',
       'lag_30_windowed_resp_class_c', 'lag_1_commercial_streak_llm',
       'lag_2_commercial_streak_llm', 'lag_3_commercial_streak_llm',
       'lag_1_sports_broadcasting_streak_llm',
       'lag_2_sports_broadcasting_streak_llm',
```

```
       'lag_3_sports_broadcasting_streak_llm', 'commercial_streak_decay',
       'sports_broadcasting_streak_decay', 'lag_1_c_decay', 'lag_1_sb_decay',
       'char_len', 'c_blocks_so_far_squared', 'sb_blocks_so_far_squared',
       'char_len_squared', 'c_blocks_so_far_root', 'sb_blocks_so_far_root',
       'char_len_root'],
      dtype='object')
Index(['sports_broadcasting_streak_llm', 'commercial_streak_llm',
       'c_blocks_so_far_llm', 'sb_blocks_so_far_llm',
       'num_blocks_since_sb_llm', 'num_blocks_since_c_llm',
       'windowed_resp_class_c', 'sequence_num', 'response_classification_c',
       'min_3_response_classification_c', 'min_10_response_classification_c',
       'min_20_response_classification_c', 'min_30_response_classification_c',
       'min_3_windowed_resp_class_c', 'min_10_windowed_resp_class_c',
       'min_20_windowed_resp_class_c', 'min_30_windowed_resp_class_c',
       'mean_3_response_classification_c', 'mean_10_response_classification_c',
       'mean_20_response_classification_c',
       'mean_30_response_classification_c', 'mean_3_windowed_resp_class_c',
       'mean_10_windowed_resp_class_c', 'mean_20_windowed_resp_class_c',
       'mean_30_windowed_resp_class_c', 'lag_1_response_classification_c',
       'lag_2_response_classification_c', 'lag_3_response_classification_c',
       'lag_4_response_classification_c', 'lag_5_response_classification_c',
       'lag_10_response_classification_c', 'lag_20_response_classification_c',
       'lag_30_response_classification_c', 'lag_1_windowed_resp_class_c',
       'lag_2_windowed_resp_class_c', 'lag_3_windowed_resp_class_c',
       'lag_4_windowed_resp_class_c', 'lag_5_windowed_resp_class_c',
       'lag_10_windowed_resp_class_c', 'lag_20_windowed_resp_class_c',
       'lag_30_windowed_resp_class_c', 'lag_1_commercial_streak_llm',
       'lag_2_commercial_streak_llm', 'lag_3_commercial_streak_llm',
       'lag_1_sports_broadcasting_streak_llm',
       'lag_2_sports_broadcasting_streak_llm',
       'lag_3_sports_broadcasting_streak_llm', 'commercial_streak_decay',
       'sports_broadcasting_streak_decay', 'lag_1_c_decay', 'lag_1_sb_decay',
       'char_len', 'c_blocks_so_far_squared', 'sb_blocks_so_far_squared',
       'char_len_squared', 'c_blocks_so_far_root', 'sb_blocks_so_far_root',
       'char_len_root'],
      dtype='object')
```

Do feature selection OR just prepare your own selection of features

In [42]:

```python
def get_feat_manually(run_mode):

    match RUN_MODE:
        case 'C':
            selected_cols = ['response_classification_c']
        case 'D':
            selected_cols = ['response_classification_c']
        case 'E':
            selected_cols = ['response_classification_c']
        case 'F':
            selected_cols = ['response_classification_c']
        case 'G':
            selected_cols = ['num_blocks_since_c_llm', 'sequence_num',
           'mean_10_response_classification_c',
           'mean_30_response_classification_c', 'mean_20_windowed_resp_class_c',
```

```
            'mean_30_windowed_resp_class_c', 'lag_1_windowed_resp_class_c',
            'lag_2_sports_broadcasting_streak_llm', 'commercial_streak_decay',
            'c_blocks_so_far_root']
        case 'H':
            selected_cols = ['response_classification_c']
        case _:
            selected_cols = ['response_classification_c']

    return selected_cols
```

In [43]:
```
# Change AUTO_SELECT_FEAT to False and select the feature list int he 'else' statem
if AUTO_SELECT_FEAT:
    selected_indices, selected_columns, X_train_selected, X_test_selected = get_fea
else:
    # selecte features mannaully here based on RUN_MODE
    # NOTE: These hard-coded feature lists are based on previous past runs of SFS
    selected_columns = get_feat_manually(RUN_MODE)

if selected_columns is not None:
    X_train_selected = X_train[selected_columns].to_numpy()
    X_test_selected = X_test[selected_columns].to_numpy()
else:
    # or just use all of them
    X_train_selected = X_train.to_numpy()
    X_test_selected = X_test.to_numpy()

print('Will run the main model with these features')
print(X_train[selected_columns].columns)
```

```
Selected features indices: [ 5  7 18 20 23 24 33 45 47 55]
Index(['num_blocks_since_c_llm', 'sequence_num',
       'mean_10_response_classification_c',
       'mean_30_response_classification_c', 'mean_20_windowed_resp_class_c',
       'mean_30_windowed_resp_class_c', 'lag_1_windowed_resp_class_c',
       'lag_2_sports_broadcasting_streak_llm', 'commercial_streak_decay',
       'c_blocks_so_far_root'],
      dtype='object')
Will run the main model with these features
Index(['num_blocks_since_c_llm', 'sequence_num',
       'mean_10_response_classification_c',
       'mean_30_response_classification_c', 'mean_20_windowed_resp_class_c',
       'mean_30_windowed_resp_class_c', 'lag_1_windowed_resp_class_c',
       'lag_2_sports_broadcasting_streak_llm', 'commercial_streak_decay',
       'c_blocks_so_far_root'],
      dtype='object')
```

```
C:\Users\bbfor\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\bas
e.py:486: UserWarning: X has feature names, but SequentialFeatureSelector was fitted
without feature names
  warnings.warn(
C:\Users\bbfor\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\bas
e.py:486: UserWarning: X has feature names, but SequentialFeatureSelector was fitted
without feature names
  warnings.warn(
```

Checking for NaNs in our data AFTER feature engineering

Note that the NaNs come from lag columns that we engineered because they can't get values prior to the beginning of the data sets

In [44]:
```
# count number of rows that have any NANs in any column
def count_nan_rows(df):
  """Counts the number of rows in a DataFrame that contain any NaN values."""
  return df.isna().any(axis=1).sum()

print("NaN check AFTER FEATURE ENGINEERING")
train_nan_count = count_nan_rows( pd.DataFrame(X_train_selected))
print(f"Train set Nans:{train_nan_count}")
test_nan_count = count_nan_rows( pd.DataFrame(X_test_selected))
print(f"Train set Nans:{test_nan_count}")

train_tot_count = len(X_train_selected)
test_tot_count = len(X_test_selected)

train_nan_frac = train_nan_count/train_tot_count
print(f"% of training rows with at least one NaN: {train_nan_frac:.2f}")

test_nan_frac = test_nan_count/test_tot_count
print(f"% of test rows with at least one NaN: {test_nan_frac:.2f}")

if test_nan_frac <= NAN_TOLERANCE and train_nan_frac <= NAN_TOLERANCE:
    print("Leaving NaNs in the data sets because small number of them and XGBoost c
else:
    raise Exception("Too many NaNs")
```

```
NaN check AFTER FEATURE ENGINEERING
Train set Nans:87
Train set Nans:29
% of training rows with at least one NaN: 0.01
% of test rows with at least one NaN: 0.02
Leaving NaNs in the data sets because small number of them and XGBoost can handle th
em
```

Now tune the model using Grid Search CV

Note: use timeseries split in cross validation to avoid data leakage

In [45]:
```
# Define a Standard Scaler to normalize inputs
scaler = StandardScaler()

tune_model_instance = xgb.XGBClassifier()
pipe = Pipeline(steps=[("scaler", scaler), ("xgb", tune_model_instance)])
param_grid = {
            'xgb__n_estimators': [5,15,20,30,40,100],
            'xgb__max_depth': [1, 2, 3, 4, 5, 7, 10, 15, 20],
            'xgb__learning_rate': [.05, 0.1, 0.15, 0.2,0.25,0.3, 0.35,0.4, 0.45,0.5
}
xgb_tscv = TimeSeriesSplit()
```

```
search = GridSearchCV(
            estimator = pipe,
            param_grid = param_grid,
            n_jobs = -1,
            cv = xgb_tscv,
            verbose = 1,
            return_train_score=True,
            scoring='f1')

search.fit(X_train_selected, y_train)
print("Best parameter (CV score=%0.3f):" % search.best_score_)
print(search.best_params_)

# map the best parameters back a params object
# return that to be used for a final training by calling function
best_n_estimators_binary= search.best_params_['xgb__n_estimators']
best_max_depth_binary= search.best_params_['xgb__max_depth']
best_learning_rate_binary= search.best_params_['xgb__learning_rate']
```

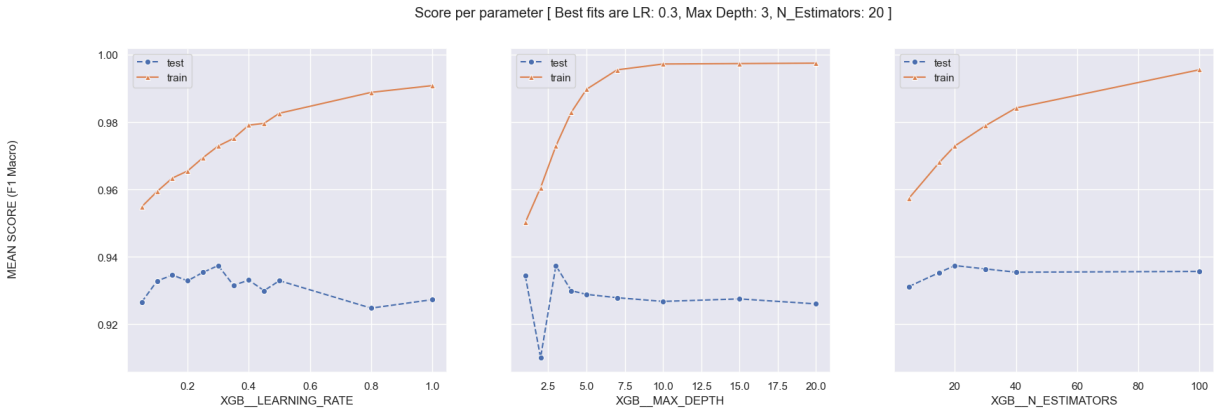
```
Fitting 5 folds for each of 648 candidates, totalling 3240 fits
Best parameter (CV score=0.937):
{'xgb__learning_rate': 0.3, 'xgb__max_depth': 3, 'xgb__n_estimators': 20}
```

In [46]:

```
def plot_search_results(grid):
    """
    Params:
        grid: A trained GridSearchCV object.
    """
    ## Results from grid search
    results = grid.cv_results_
    means_test = results['mean_test_score']
    means_train = results['mean_train_score']

    ## Getting indexes of values per hyper-parameter
    masks=[]
    masks_names= list(grid.best_params_.keys())
    for p_k, p_v in grid.best_params_.items():
        masks.append(list(results['param_'+p_k].data==p_v))

    params=grid.param_grid

    ## Ploting results
    fig, ax = plt.subplots(1,len(params),sharex='none', sharey='all',figsize=(2*FIG
    fig.suptitle(f"Score per parameter [ Best fits are LR: {best_learning_rate_bina
    fig.text(0.04, 0.5, f"MEAN SCORE (F1 Macro)" , va='center', rotation='vertical'
    pram_preformace_in_best = {}
    for i, p in enumerate(masks_names):
        m = np.stack(masks[:i] + masks[i+1:])
        pram_preformace_in_best
        best_parms_mask = m.all(axis=0)
        best_index = np.where(best_parms_mask)[0]
        x = np.array(params[p])
        y_1 = np.array(means_test[best_index])
        y_2 = np.array(means_train[best_index])
        sns.lineplot(x = x, y=y_1, label = 'test', ax=ax[i], linestyle='--', marker
        sns.lineplot(x = x, y=y_2, label = 'train', ax=ax[i], linestyle='-', marker
```

```
        ax[i].set_xlabel(p.upper())

    plt.legend()
    plt.show()
```

In [47]:
```
plot_search_results(search)
```



Grid Search helped us find parameters that optimize for validation sets.

Charts show F1 for each parameter (holding other parameters to their 'best' fit).

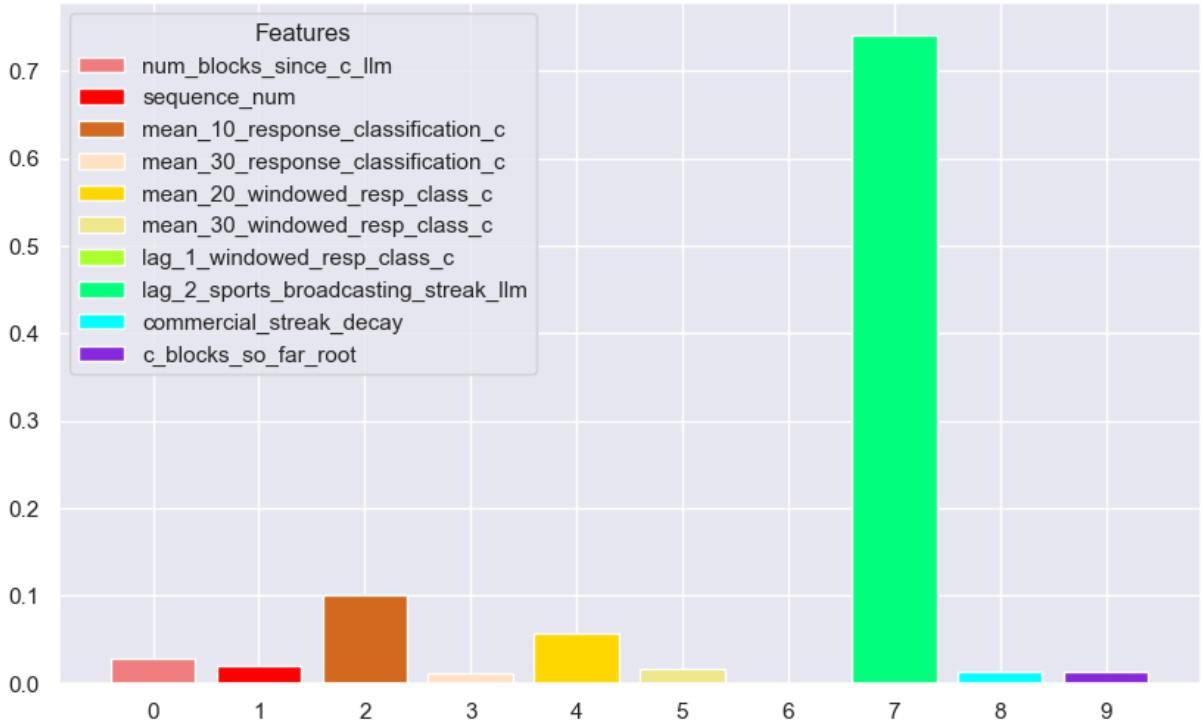
We are NOT overfitting because we chose highest for validation set at the point of divergence from training curve

In [48]:
```
# finally run for all training data with best parameters we found and the best feat

scaler = StandardScaler()

# Fit the scaler and transform the features
X_train_scaled = scaler.fit_transform(X_train_selected, y_train)

# instantiate model we will use to search for best features
final_model_instance = xgb.XGBClassifier(n_estimators=best_n_estimators_binary, max

# Train the XGBoost model on the selected features
final_model_instance.fit(X_train_scaled, y_train)

# Make predictions and evaluate the model
X_test_scaled = scaler.transform(X_test_selected)
y_pred = final_model_instance.predict(X_test_scaled)
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred, average = 'macro')
print("Raw accuracy with selected features:", accuracy)
print("F1 with selected features:", f1)

# feature colors (assumes no more than 10 features)
impt_colors = ['lightcoral', 'red', 'chocolate','bisque','gold', 'khaki','greenyell
```

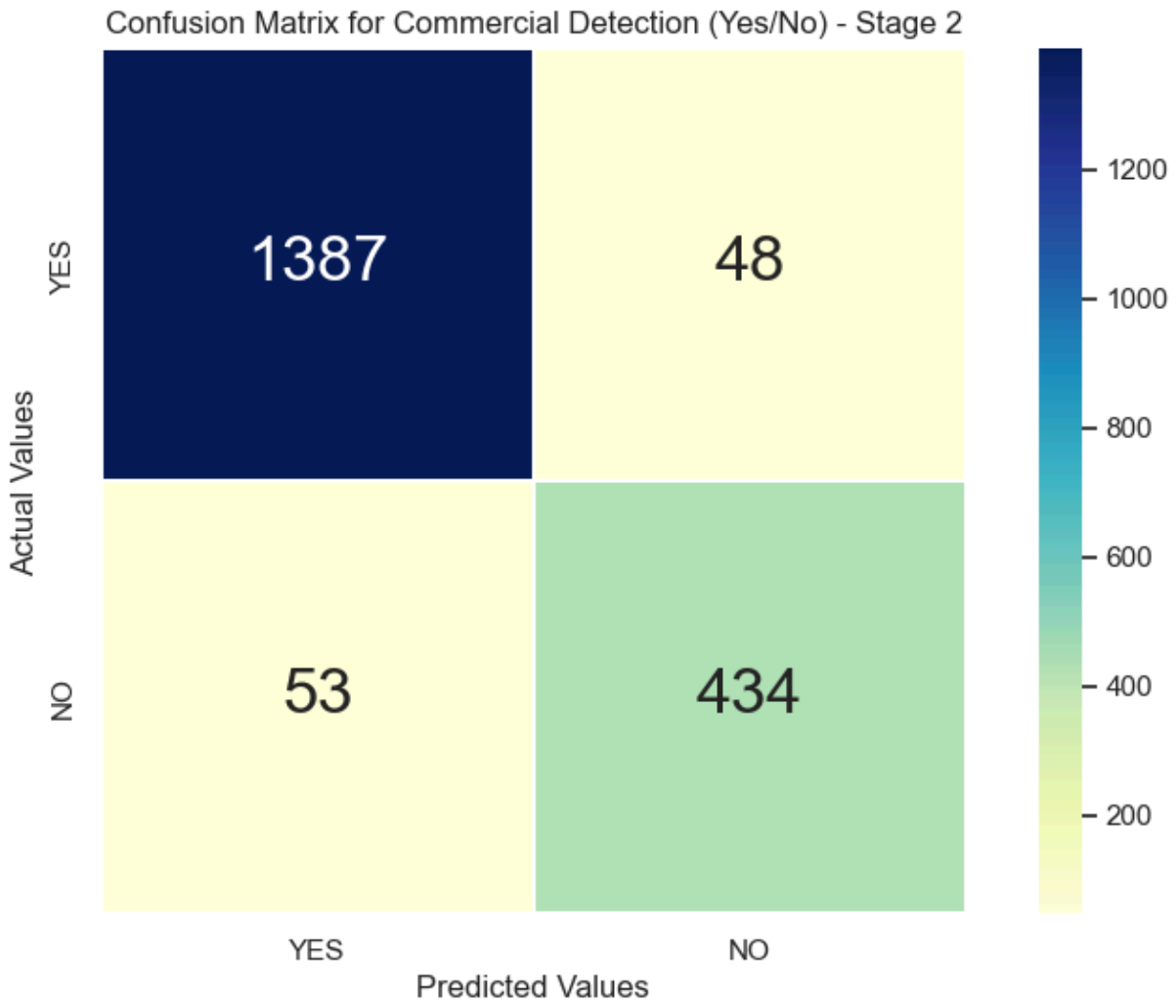
```
plt.figure(figsize=(FIGX,FIGY))
plt.bar(range(len(final_model_instance.feature_importances_)), final_model_instance
plt.xticks(range(len(final_model_instance.feature_importances_)))
plt.legend(title='Features')  # Add legend with title
```

```
Raw accuracy with selected features: 0.9474505723204995
F1 with selected features: 0.9303191995333603
```

Out[48]: <matplotlib.legend.Legend at 0x17a67d7e410>



In [49]:
```
eval_accuracy( 'XGBoost', 'Stage 2', y_test,y_pred)
```




```
recall_score_macro_Stage 2 : 0.9289
precision_score_macro_Stage 2 : 0.9318
f1_score_macro_Stage 2 : 0.9303
bal_acc_score_Stage 2 : 0.9289
<Figure size 640x480 with 0 Axes>
```

In [50]:
```
df_y_pred = pd.DataFrame(y_pred)
df_y_pred.head()
first_column_name = df_y_pred.columns[0]
df_y_pred = df_y_pred.rename(columns={first_column_name: 'stage2_response_classific
df_y_pred.head()
```

Out[50]:

	stage2_response_classification_c
0	0
1	0
2	1
3	1
4	1

In [ ]:

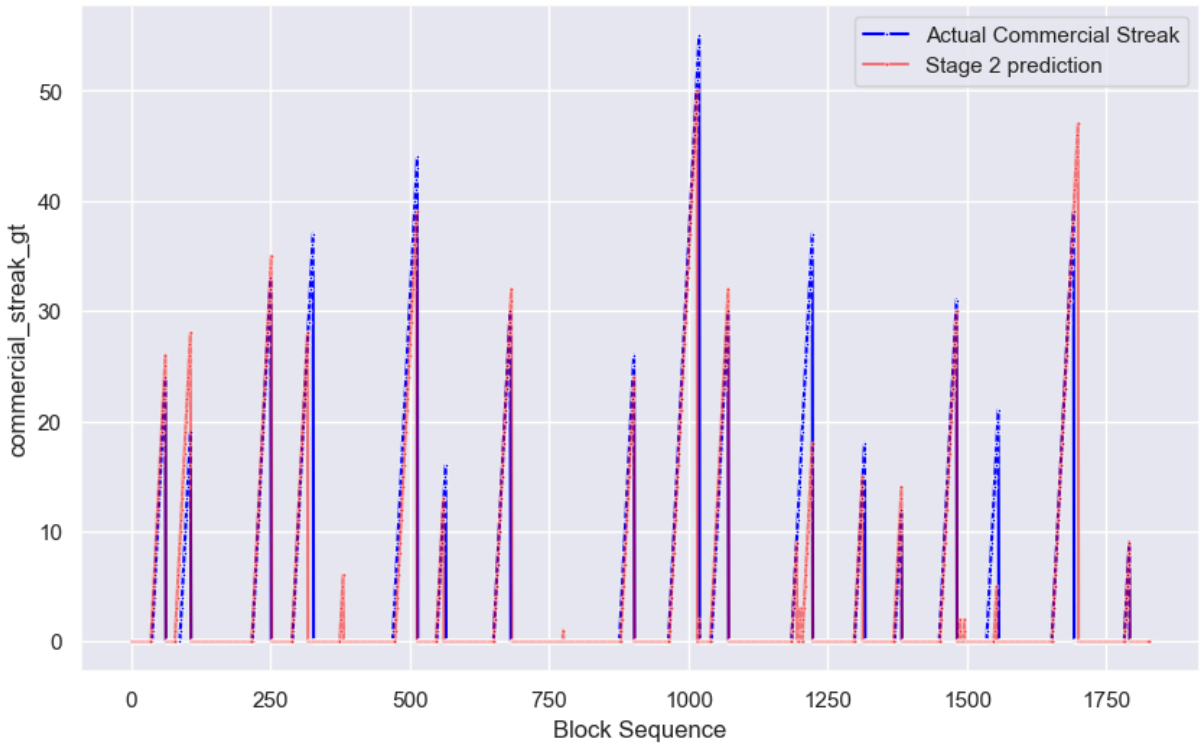
In [51]:
```
# construc a dataframe to do reporitng
def construct_df_reporting(X_test, df_test_all_cols, y_pred, filename_prefix=''):
    # all the x test features and original columns
    df_test_all_cols_wo_seq = df_test_all_cols.drop('sequence_num', axis=1)
    df_test_results_1 = pd.concat([X_test,df_test_all_cols_wo_seq], axis=1)
    df_tr_re = df_test_results_1.reset_index()
    df_tr_re.to_csv('df_tr_re.csv')

    # add on predictions from stage 2
    df_y_pred_re = df_y_pred.reset_index()
    df_test_results_all = pd.concat([df_tr_re,df_y_pred_re], axis=1)
    df_test_results_all.to_csv('df_test_results_all.csv')

    # create stage 2 stat columns
    df_test_results_all['stage2_sports_broadcasting_streak'] = 0
    df_test_results_all['stage2_num_blocks_since_sb'] = 0
    df_test_results_all['stage2_commercial_streak'] = 0
    df_test_results_all['stage2_num_blocks_since_c'] = 0
    df_test_results_all['stage2_sb_blocks_so_far'] = 0
    df_test_results_all['stage2_c_blocks_so_far'] = 0

    # calculate streaks based off stage 2
    stat_util = LLMStatsUtils()
    df_test_results_w_stats = stat_util.update_llm_stats(df_test_results_all, 'stag
    df_test_results_w_stats.to_csv(filename_prefix + 'df_test_results_w_stats.csv')

    return df_test_results_w_stats


df_test_results_w_stats = construct_df_reporting(X_test, df_test_all_cols, y_pred)
```

New session

In [ ]:

In [52]:
```
def chart_stage2_test_y_vs_sequence_p_and_gt(df_stage2_test, title_str, y1_field_na
    df_len = len(df_stage2_test)
    df_start = int(round(df_len*start_at_pct,0))
    df_end = int(round(df_len*end_at_pct,0))

    df_filtered = df_stage2_test.iloc[df_start:df_end]
    # Set the aesthetic style of the plots
    sns.set(style="darkgrid")
    plt.figure(figsize=(FIGX, FIGY))
    sns.lineplot(x='sequence_num', y=y1_field_name, data=df_filtered, marker='o',
    sns.lineplot(x='sequence_num', y=y2_field_name, data=df_filtered, marker='o',
    # Add title and labr)
    plt.xlabel('Block Sequence')
    #plt.ylabel(y_field_title)

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

In [53]:
```
chart_stage2_test_y_vs_sequence_p_and_gt(df_test_results_w_stats, 'Stage 2 Commerci
```



In [54]:
```
# Assume we have populated df_train_lr and df_test_lr earlier

df_train_lr_wo_nans = df_train_lr.dropna()
df_test_lr_wo_nans = df_test_lr.dropna()

X_train_lr, y_train_lr, X_test_lr, y_test_lr = get_X_y(df_train_lr_wo_nans, df_test
```

In [55]:
```
if selected_columns is not None:
    X_train_selected_lr = X_train_lr[selected_columns].to_numpy()
    X_test_selected_lr = X_test_lr[selected_columns].to_numpy()
else:
    # or just use all of them
    X_train_selected_lr = X_train_lr.to_numpy()
    X_test_selected_lr = X_test_lr.to_numpy()
```

In [ ]:
```
```

In [56]:
```
# LOGISTIC REGRESSION CLASSIFICATION MODEL
# Define a Standard Scaler to normalize inputs
scaler_lr = StandardScaler()
# instantiate model
model_instance_lr = LogisticRegression(solver='saga',max_iter = 800, tol=0.1)
pipe = Pipeline(steps=[("scaler", scaler_lr), ("logistic", model_instance_lr)])
param_grid = {
                'logistic__penalty': ['l1', 'l2'],
                'logistic__C': [0.001, 0.01, 0.1, 1, 10, 100]}
lr_tscv = TimeSeriesSplit()
```

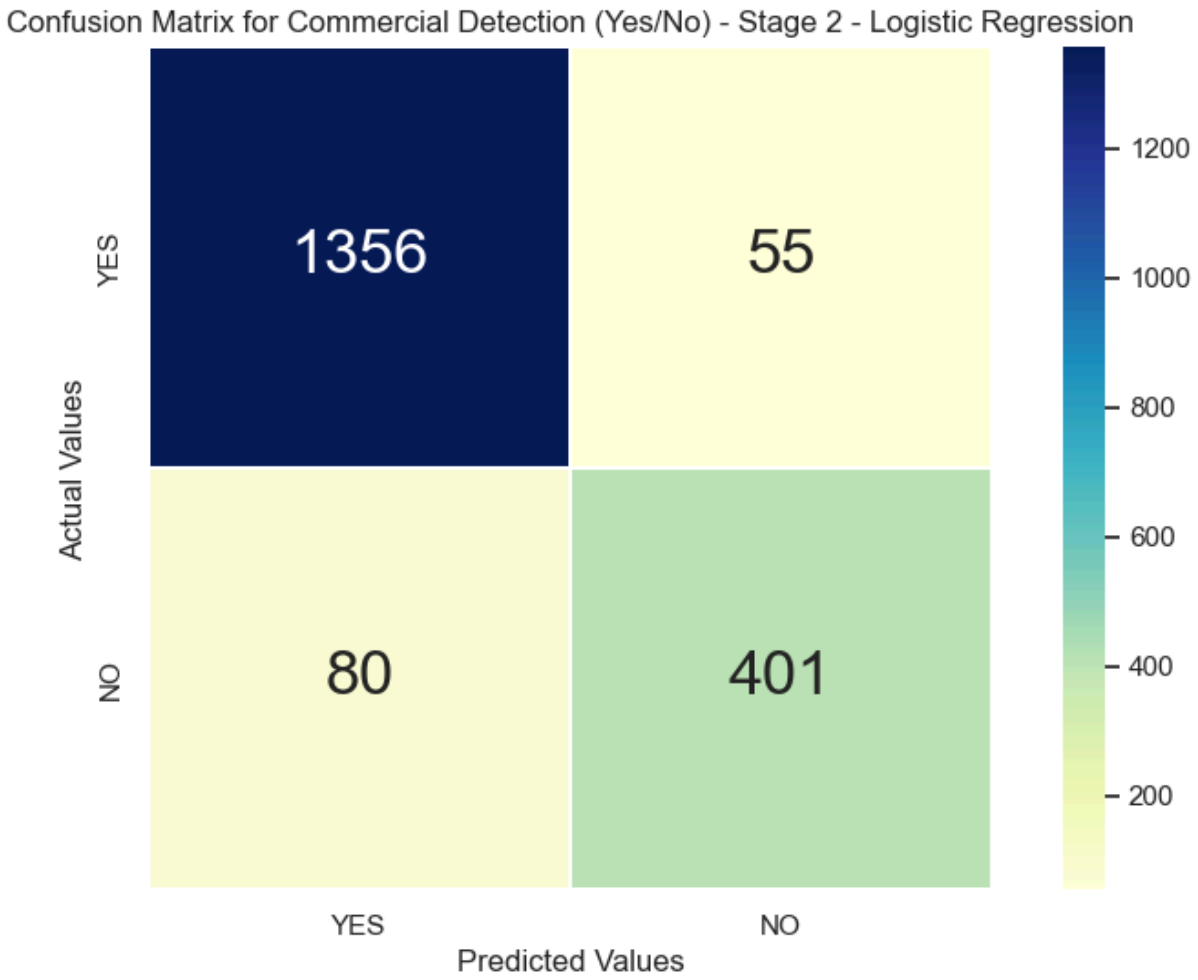
```
search_lr = GridSearchCV(
            estimator = pipe,
            param_grid = param_grid,
            n_jobs = -1,
            cv = lr_tscv,
            verbose = 1,
            return_train_score=True,
            scoring='f1')

search_lr.fit(X_train_selected_lr, y_train_lr)
print("Best parameter (CV score=%0.3f):" % search_lr.best_score_)
print(search_lr.best_params_)

# map the best parameters back a params object
# return that to be used for a final training
best_penalty_binary= search_lr.best_params_['logistic__penalty']
best_C_binary= search_lr.best_params_['logistic__C']
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
Best parameter (CV score=0.893):
{'logistic__C': 0.1, 'logistic__penalty': 'l1'}
```

In [57]:
```
# Now run LR model for all training data with best parameters we found and the best
scaler_lr = StandardScaler()

# Fit the scaler and transform the features
X_train_scaled_lr = scaler_lr.fit_transform(X_train_selected_lr, y_train_lr)

# instantiate model we will use to search for best features
final_model_instance_lr = LogisticRegression(solver='saga',max_iter = 800, tol=0.1,

# Train the XGBoost model on the selected features
final_model_instance_lr.fit(X_train_scaled_lr, y_train_lr)

# Make predictions and evaluate the model
X_test_scaled_lr = scaler_lr.transform(X_test_selected_lr)
y_pred_lr = final_model_instance_lr.predict(X_test_scaled_lr)
accuracy_lr = accuracy_score(y_test_lr, y_pred_lr)
f1_lr = f1_score(y_test_lr, y_pred_lr, average='macro')
print("Raw Accuracy with selected features:", accuracy_lr)
print("F1 with selected features:", f1_lr)
importance = permutation_importance(final_model_instance_lr, X_test_selected_lr, y_
```

```
Raw Accuracy with selected features: 0.928646934460888
F1 with selected features: 0.9042524119642875
```

In [58]:
```
eval_accuracy( 'Logistic Regression', 'Stage 2 - Logistic Regression', y_test_lr,y_
```




```
recall_score_macro_Stage 2 - Logistic Regression : 0.8974
precision_score_macro_Stage 2 - Logistic Regression : 0.9118
f1_score_macro_Stage 2 - Logistic Regression : 0.9043
bal_acc_score_Stage 2 - Logistic Regression : 0.8974
<Figure size 640x480 with 0 Axes>
```

In [ ]:

In [ ]: