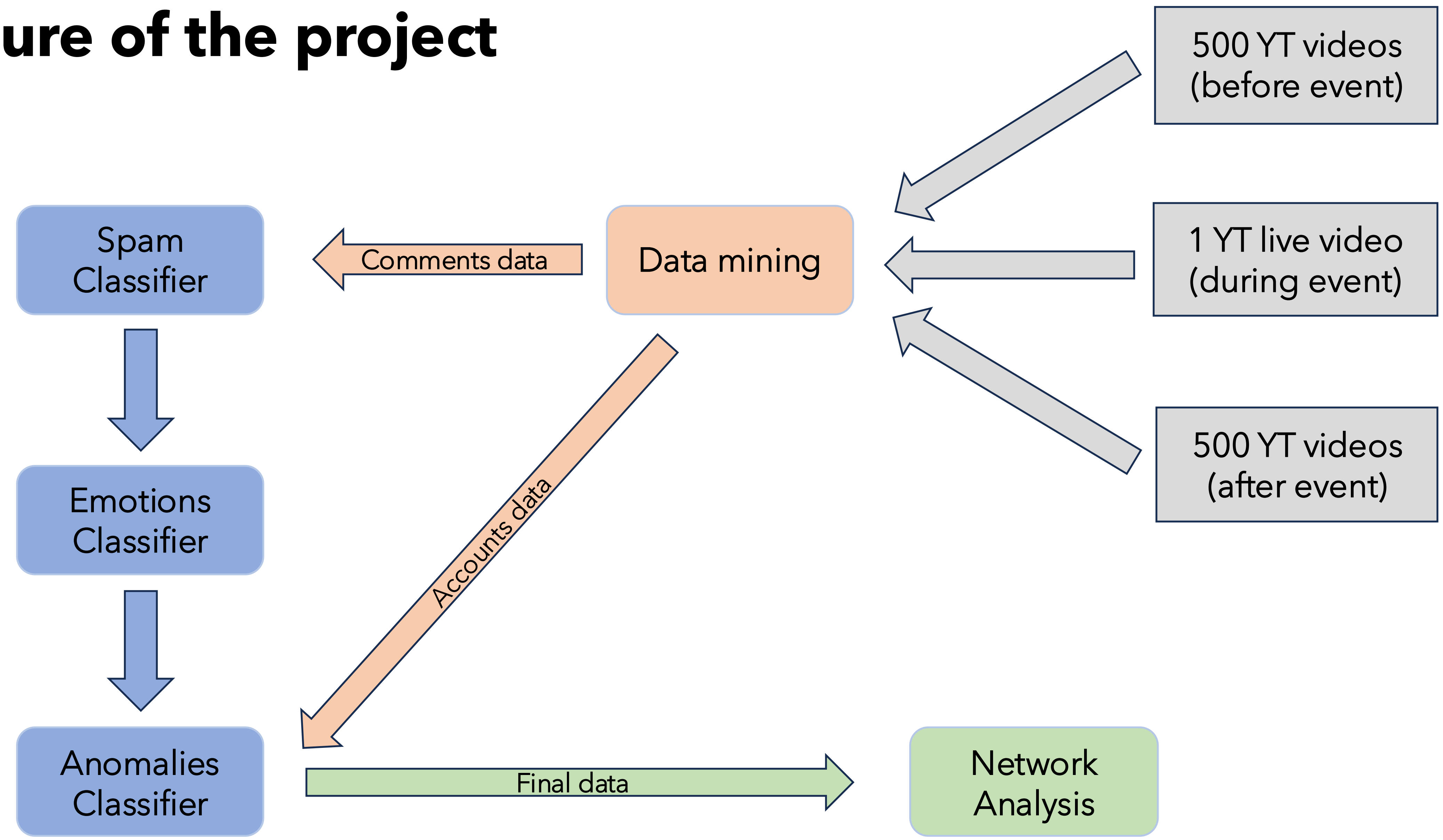
Social Media Mining Project

OBJECTIVE: ANALYZING SHIFTS IN USER BEHAVIOR AND EMOTIONS BEFORE, DURING, AND AFTER THE IPHONE 16 ANOUNCEMENT ON SEPTEMBER 9, 2024

Structure of the project



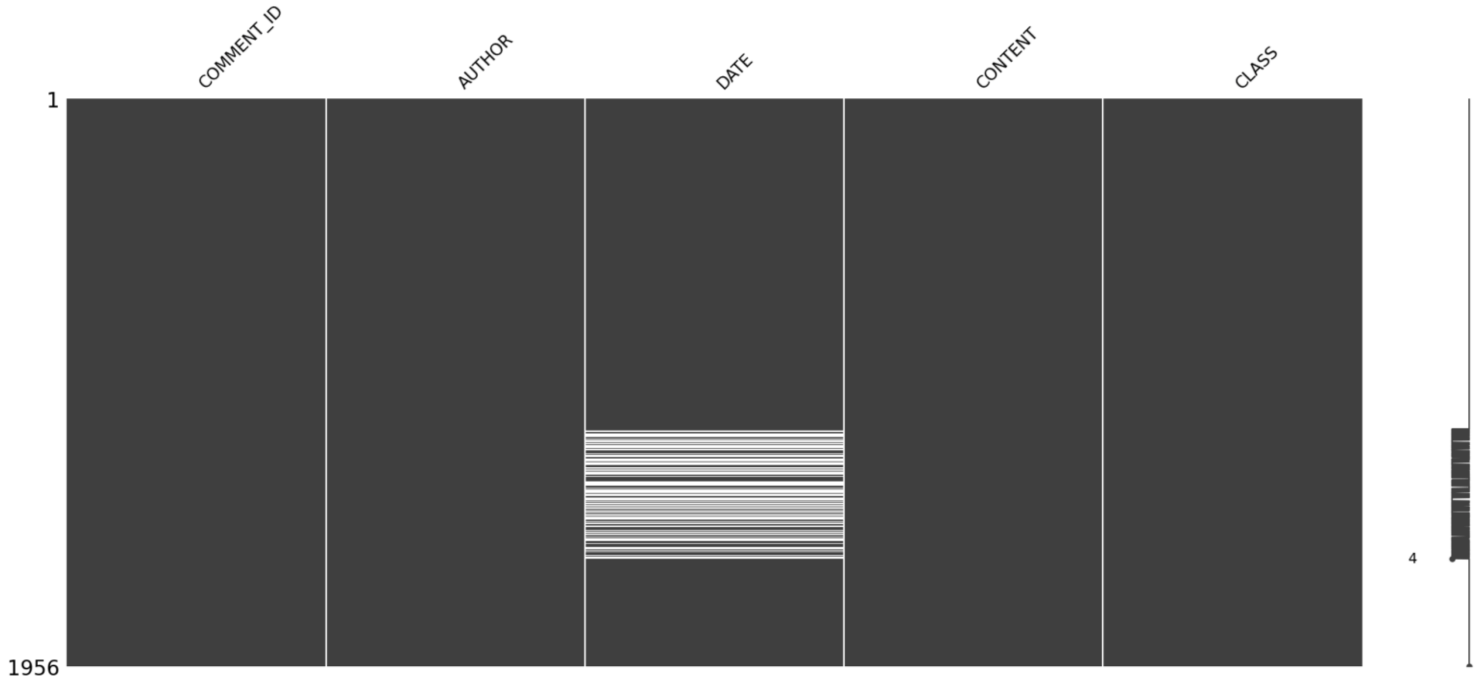
Research questions

- What words are most associated with the topic? Can we identify iphone 16 features from the most used ones?
- What sentiment is associated with iphone 16 features?
- Have feature sentiments shifted before, during and after the apple event?
- Has spam comments density shifted before, during and after the apple event?
- What are the accounts that produce the most spam?
- What percentage of anomalous accounts generate spam?
- Are the nodes central because they produce spam or because of engagement?
- Are spam generator accounts likely to comment on the same videos (target similar videos)? What about anomalous accounts? what about accounts with similar sentiment behaviour?
- Is the spam percentage in the biggest communities higher than the general spam percentage?
- How is the sentiment distributed across the biggest communities?

Spam Classifier – Downloaded dataset

```
msno.matrix(original_df)
```

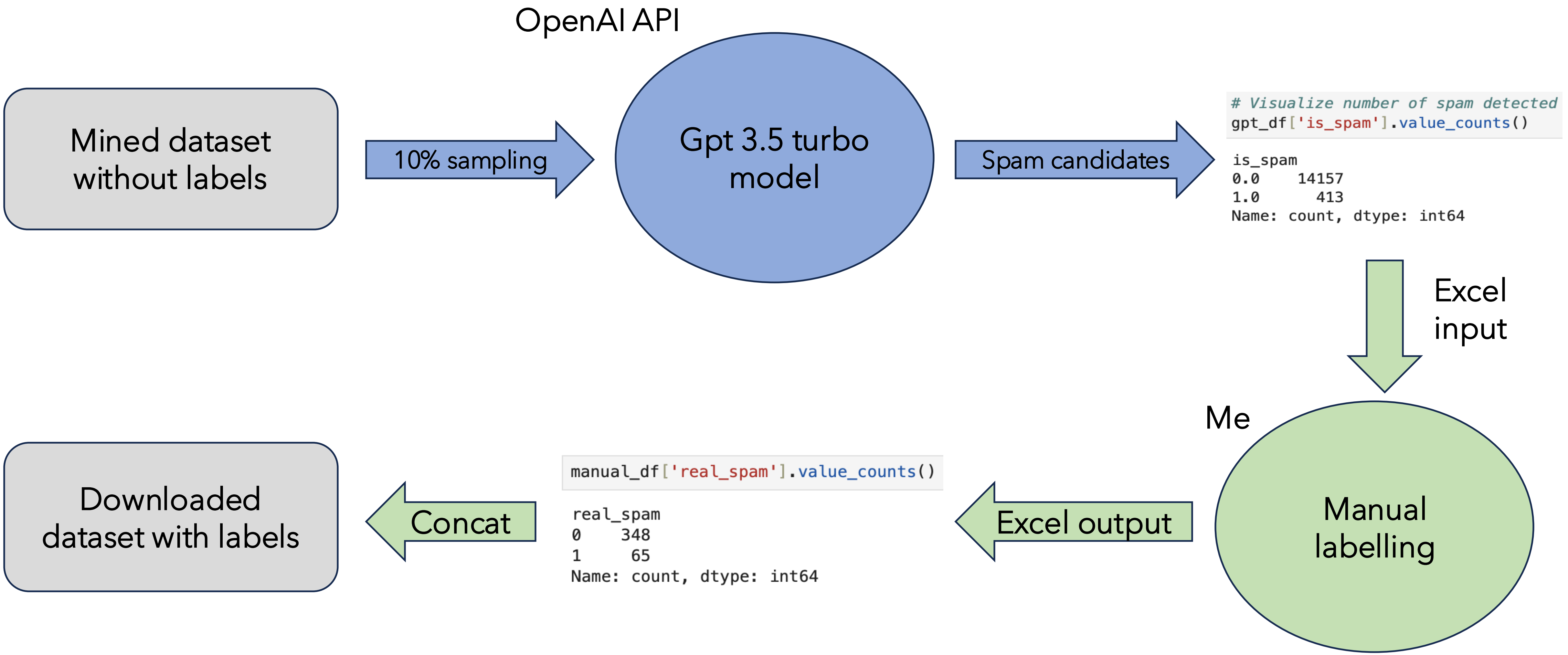
<Axes: >




```
print(original_df['CLASS'].value_counts())
```

```
CLASS
1    1005
0     951
Name: count, dtype: int64
```

Spam Classifier – Semi-Supervised Learning Labelling



Spam Classifier – Custom Transformers

Data Cleaner Custom Transformer

```python
class DataCleaner(BaseEstimator, TransformerMixin):
    def __init__(self):
        DetectorFactory.seed = 0 # Make language identification consistent if performed multiple times

    def fit(self, X, y=None):
        return self

    def transform(self, X, y=None):
        df = pd.DataFrame(X)

        # Remove non english comments
        df['language'] = df['content'].apply(self.detect_language)
        df = df[df['language'] == 'en']
        df.drop(['language'], axis=1, inplace=True)

        return df

    def detect_language(self, text):
        try:
            return detect(text)
        except LangDetectException:
            return 'unknown'
```

Spam Classifier – Custom Transformers

Feature Extraction Custom Transformer

```
class FeatureExtractor(BaseEstimator, TransformerMixin):
    def __init__(self):
        self.sia = SentimentIntensityAnalyzer()

    def fit(self, X, y=None):
        return self

    def transform(self, X, y=None):
        # X is expected to be a pandas Series of text comments (i.e., the 'content' column)
        df = pd.DataFrame(X)
        df['text_length'] = df['content'].apply(len)
        df['num_links'] = df['content'].apply(lambda x: len(re.findall(r'http[s]?://\S+', x)))
        df['num_special_chars'] = df['content'].apply(lambda x: len(re.findall(r'[^a-zA-Z0-9\s]', x)))
        df['capitalization_ratio'] = df['content'].apply(lambda x: sum(1 for c in x if c.isupper()) / (len(x) + 1))  # Avoid division by zero
        df['num_digits'] = df['content'].apply(lambda x: sum(c.isdigit() for c in x))
        df['num_repeated_chars'] = df['content'].apply(lambda x: sum([1 for i in range(1, len(x)) if x[i] == x[i-1]]))

        # Sentiment score
        df['sentiment_score'] = df['content'].apply(lambda x: self.sia.polarity_scores(x)['compound'])

        return df
```

Spam Classifier – Custom Transformers

Text Preprocessing Custom Transformer

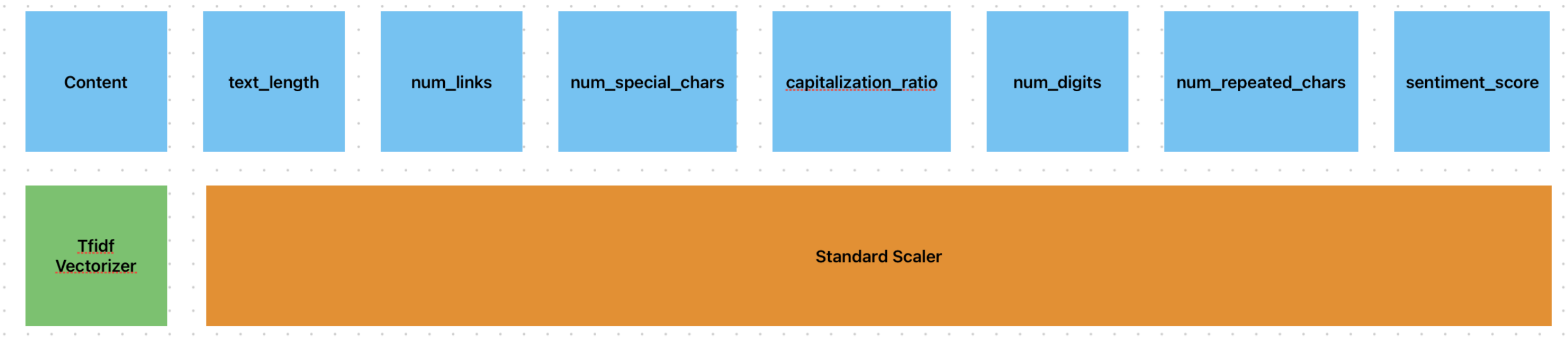
```python
class TextPreprocessingTransformer(BaseEstimator, TransformerMixin):
    def __init__(self):
        # Load resources in the constructor
        self.stop_words = set(stopwords.words('english'))
        self.lemmatizer = WordNetLemmatizer()

    def fit(self, X, y=None):
        return self

    def transform(self, X, y=None):
        # Apply text preprocessing
        X['content'] = X['content'].apply(self.preprocess_text)
        return X
```

```python
    def preprocess_text(self, text):
        # Function to reduce repeated characters (e.g., "helooo" -> "hello")
        def reduce_repeated_characters(word):
            return re.sub(r'(.)\1+', r'\1\1', word)

        # 1. Lowercase the text
        text = text.lower()

        # 2. Remove URLs
        text = re.sub(r'http[s]?://\S+', '', text)

        # 3. Remove special characters, numbers, and punctuation
        text = re.sub(r'[^a-zA-Z\s]', '', text)

        # 4. Remove stopwords
        words = text.split()
        words = [word for word in words if word not in self.stop_words]

        # 5. Reduce repeated characters (e.g., "helooo" -> "hello")
        words = [reduce_repeated_characters(word) for word in words]

        # 6. Lemmatization (convert words to their base form)
        words = [self.lemmatizer.lemmatize(word) for word in words]

        # 7. Join the words back into a single string
        return ' '.join(words)
```

Spam Classifier – Pipelines

```
# Data Preparation Pipeline
data_preparation_pipeline = IMBPipeline([
    ('cleaning', DataCleaner()),
    ('features_extraction', FeatureExtractor()),
    ('text_preprocessing', TextPreprocessingTransformer())
])
```

Spam Classifier – Pipelines




```python
# Model Pipeline
features_conversion = ColumnTransformer(
    transformers=[
        ('tfidf_vectorization', TfidfVectorizer(), 'content'),
        ('scaler', StandardScaler(), ['text_length', 'num_links', 'num_special_chars', 'capitalization_ratio', 'num_digits', 'num_repeated_chars', 'sentiment_score'])
    ],
    remainder='drop',
    verbose_feature_names_out=False,
    sparse_threshold=0
)

model_pipeline = IMBPipeline([
    ('features_conversion', features_conversion),
    ('sampler', SMOTE(random_state=42)),
    ('dim_reduction', PCA(n_components=0.8, random_state=42)),
    ('classifier', Perceptron(random_state=42))
])
```

Spam Classifier – Nested Stratified Cross validation

```
sampler_configs = [
    {
        'sampler': ['passthrough']  # No sampling
    },
    {
        'sampler': [SMOTE(random_state=42)],
        'sampler__sampling_strategy': ['auto']
    },
    {
        'sampler': [RandomOverSampler(random_state=42)],
        'sampler__sampling_strategy': ['minority', 'auto']
    },
    {
        'sampler': [RandomUnderSampler(random_state=42)],
        'sampler__sampling_strategy': ['majority', 0.5]
    }
]

dim_reduction_configs = [
    {
        'dim_reduction': [None]
    },
    {
        'dim_reduction': [PCA(random_state=42)],
        'dim_reduction__n_components': [0.5, 0.7, 0.9]
    }
]

classifier_configs = [
    {
        'classifier__eta0': loguniform(0.001, 100),
        'classifier': [Perceptron(random_state=42)],
        'classifier__max_iter': randint(1000, 5000),
        'classifier__class_weight': [None, 'balanced']
    },
    {
        'classifier': [LogisticRegression(solver='saga', random_state=42)],
        'classifier__C': loguniform(0.001, 100),
        'classifier__penalty': ['l1', 'l2'],
        'classifier__max_iter': randint(1000, 5000),
        'classifier__class_weight': [None, 'balanced']
    },
    {
        'classifier': [KNeighborsClassifier()],
        'classifier__n_neighbors': randint(3, 20)
    },
    {
        'classifier' : [RandomForestClassifier(random_state=42)],
        'classifier__n_estimators' : randint(10, 500)
    }
]
```

```
all_configs = []
for configuration in itertools.product(sampler_configs,dim_reduction_configs,classifier_configs):
    # Merging of three dictionary into one
    all_parameters = []
    for element in configuration:
        for item in element.items():
            all_parameters.append(item)
    all_configs.append(dict(all_parameters)) # by dict(all_parameters) we create a dict from a list of pairs (key:value)
```

Parameters

Execution

```
# Ensure class balance in training and test splits (Stratified Cross Validation)
inner_kfold = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)
outer_kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Avoid overfitting with Nested Cross Validation to have unbiased estimate of model performance
rs = RandomizedSearchCV(
    estimator = model_pipeline,
    param_distributions=all_configs,
    n_iter=len(all_configs) * 5,
    n_jobs=-1,
    cv = inner_kfold,
    scoring='f1',
    random_state = 42
)

# Perform cross-validation with stratified folds
scores = cross_validate(rs, X_train, y_train, scoring='f1', cv=outer_kfold, return_estimator=True, verbose=3)
```

Spam Classifier – Model Selection

```
passthrough
None
RandomForestClassifier(n_estimators=170, random_state=42) {'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_feature
s': 'sqrt', 'max_leaf_nodes': None, 'max_samples': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'mono
tonic_cst': None, 'n_estimators': 170, 'n_jobs': None, 'oob_score': False, 'random_state': 42, 'verbose': 0, 'warm_start': False}
f1_score: 0.9337349397590361
----------
SMOTE(random_state=42)
None
RandomForestClassifier(n_estimators=274, random_state=42) {'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_feature
s': 'sqrt', 'max_leaf_nodes': None, 'max_samples': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'mono
tonic_cst': None, 'n_estimators': 274, 'n_jobs': None, 'oob_score': False, 'random_state': 42, 'verbose': 0, 'warm_start': False}
f1_score: 0.9085365853658537
----------
SMOTE(random_state=42)
None
RandomForestClassifier(n_estimators=409, random_state=42) {'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_feature
s': 'sqrt', 'max_leaf_nodes': None, 'max_samples': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'mono
tonic_cst': None, 'n_estimators': 409, 'n_jobs': None, 'oob_score': False, 'random_state': 42, 'verbose': 0, 'warm_start': False}
f1_score: 0.9244712990936556
----------
passthrough
None
RandomForestClassifier(n_estimators=475, random_state=42) {'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_feature
s': 'sqrt', 'max_leaf_nodes': None, 'max_samples': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'mono
tonic_cst': None, 'n_estimators': 475, 'n_jobs': None, 'oob_score': False, 'random_state': 42, 'verbose': 0, 'warm_start': False}
f1_score: 0.9329268292682927
----------
RandomOverSampler(random_state=42, sampling_strategy='minority')
None
LogisticRegression(C=55.51721685244721, class_weight='balanced', max_iter=9433,
                   random_state=42, solver='saga') {'C': 55.51721685244721, 'class_weight': 'balanced', 'dual': False, 'fit_intercept': True, 'intercept_scaling': 1, 'l1_ra
tio': None, 'max_iter': 9433, 'multi_class': 'deprecated', 'n_jobs': None, 'penalty': 'l2', 'random_state': 42, 'solver': 'saga', 'tol': 0.0001, 'verbose': 0, 'warm_start':
False}
f1_score: 0.9345238095238095
----------
```

Spam Classifier – Fine tuning best model

```
F1 on training set:0.9994162288382954, F1 on test set:0.916256157635468
```

```python
# Reeplicating best model's pipeline structure
best_model_pipeline = IMBPipeline([
    ('features_conversion', features_conversion),
    ('classifier',RandomForestClassifier())
])
```

```python
# Searching for params close to the range of the best model
params = {
    'classifier__n_estimators' : randint(140, 200), # Number of trees in the forest
    'classifier__max_depth' : randint(5,30), # Controls the maximum depth of each tree. Limiting depth helps prevent overfitting
    'classifier__min_samples_split' : randint(1, 4), # The minimum number of samples required to split an internal node. Larger values prevent the model from learning overl
    'classifier__min_samples_leaf' : randint(1, 3), # The minimum number of samples required to be at a leaf node. A smaller value allows the model to capture finer details
    'classifier__max_features': ['sqrt', 'log2'], # The number of features to consider when looking for the best split. Lower values may help reduce overfitting
    'classifier__bootstrap': [True, False], # Whether samples are drawn with replacement. Bootstrapping increases model robustness
    'classifier__class_weight': [None, 'balanced'] # Handling class imbalance by adjusting the weight of each class
}
```

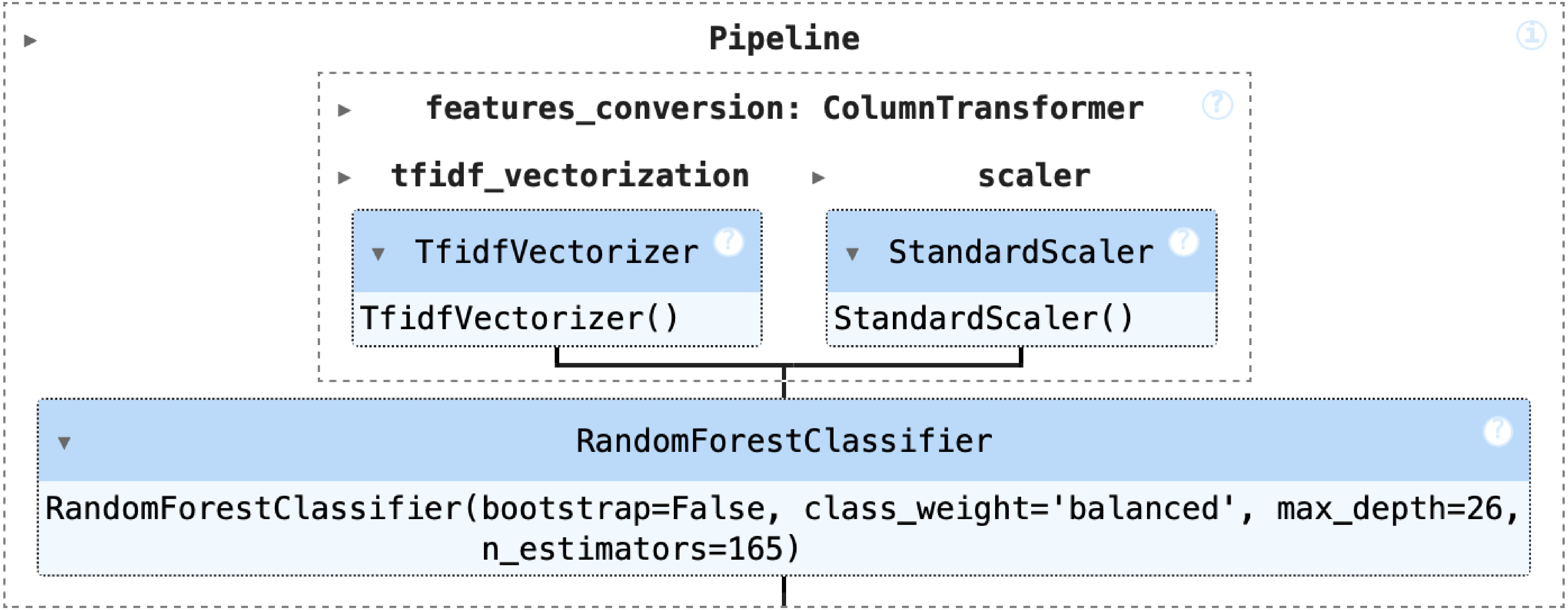
```python
rs_best = RandomizedSearchCV(
    estimator = best_model_pipeline,
    param_distributions = params,
    cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3),
    n_iter=20,
    scoring='f1',
    n_jobs=-1,
    random_state = 42
)
```

```python
rs_best.fit(X_train, y_train)
```

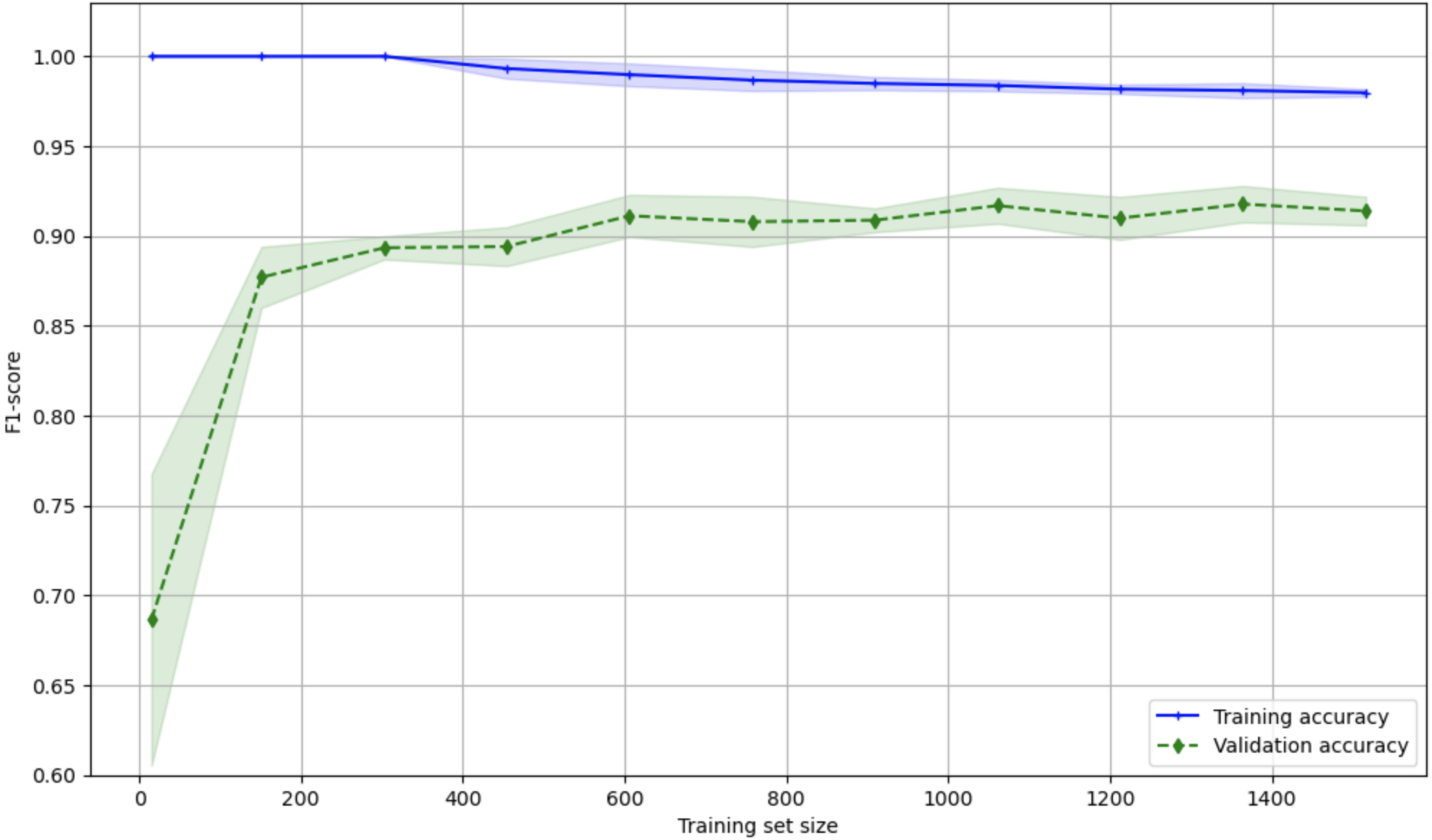
```
Selected model: F1 on training set:0.9760765550239234, F1 on test set:0.913151364764268
```

Spam Classifier – Fine tuning best model

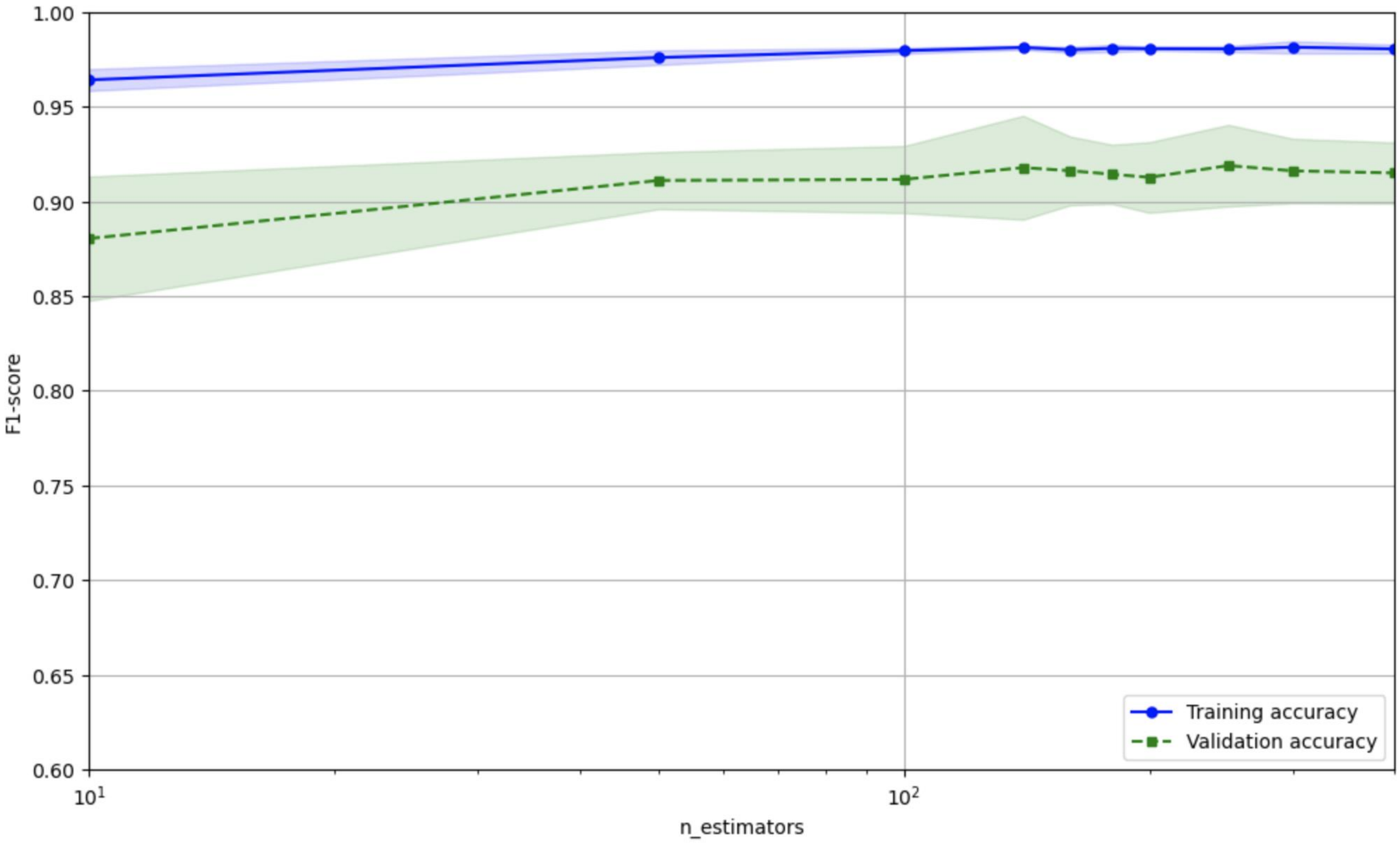
```
rs_best.best_estimator_
```



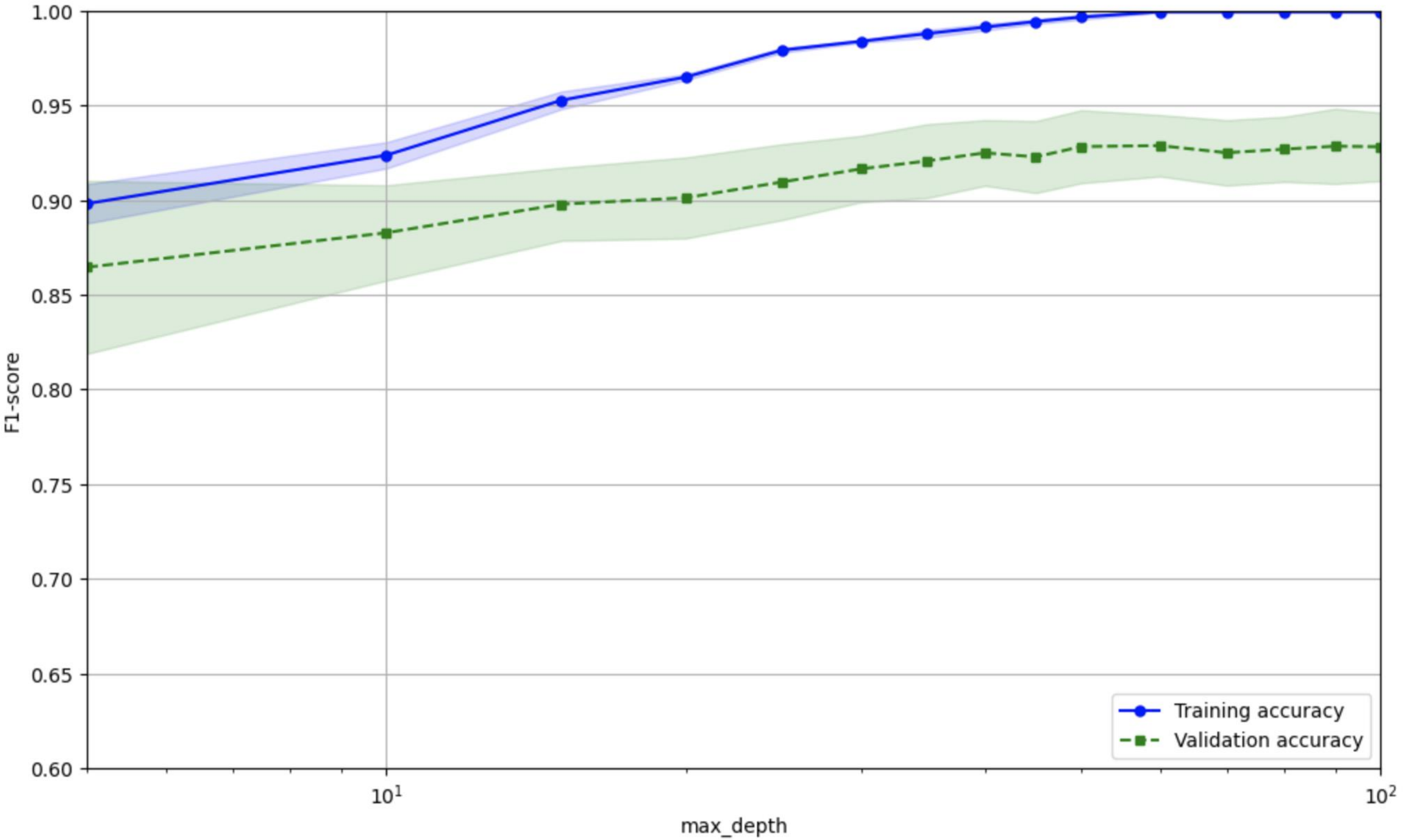
Spam Classifier – Learning curve



Spam Classifier – Validation curve



Spam Classifier – Validation curve



Spam Classifier – Predicting Spam Comments Results

RESEARCH QUESTION: Has spam comments density shifted before, during and after the apple event?

```
input1_filename = COMMENTS_BEFORE_WITH_SPAM_FILENAME
input2_filename = COMMENTS_LIVE_WITH_SPAM_FILENAME
input3_filename = COMMENTS_AFTER_WITH_SPAM_FILENAME

before_spam_df = pd.read_csv(input1_filename)
live_spam_df = pd.read_csv(input2_filename)
after_spam_df = pd.read_csv(input3_filename)

print(f'Number of spam comments before the event: {before_spam_df[before_spam_df['is_spam']==1]['is_spam'].count()} out of: {len(before_spam_df)} comments ({before_spam_df['
print(f'Number of spam comments during the event: {live_spam_df[live_spam_df['is_spam']==1]['is_spam'].count()} out of: {len(live_spam_df)} comments ({live_spam_df['is_spam'
print(f'Number of spam comments after the event: {after_spam_df[after_spam_df['is_spam']==1]['is_spam'].count()} out of: {len(after_spam_df)} comments ({after_spam_df['is_sp
```

```
Number of spam comments before the event: 4030 out of: 93301 comments (4.32%).
Number of spam comments during the event: 30 out of: 1907 comments (1.57%).
Number of spam comments after the event: 6179 out of: 131381 comments (4.70%).
```

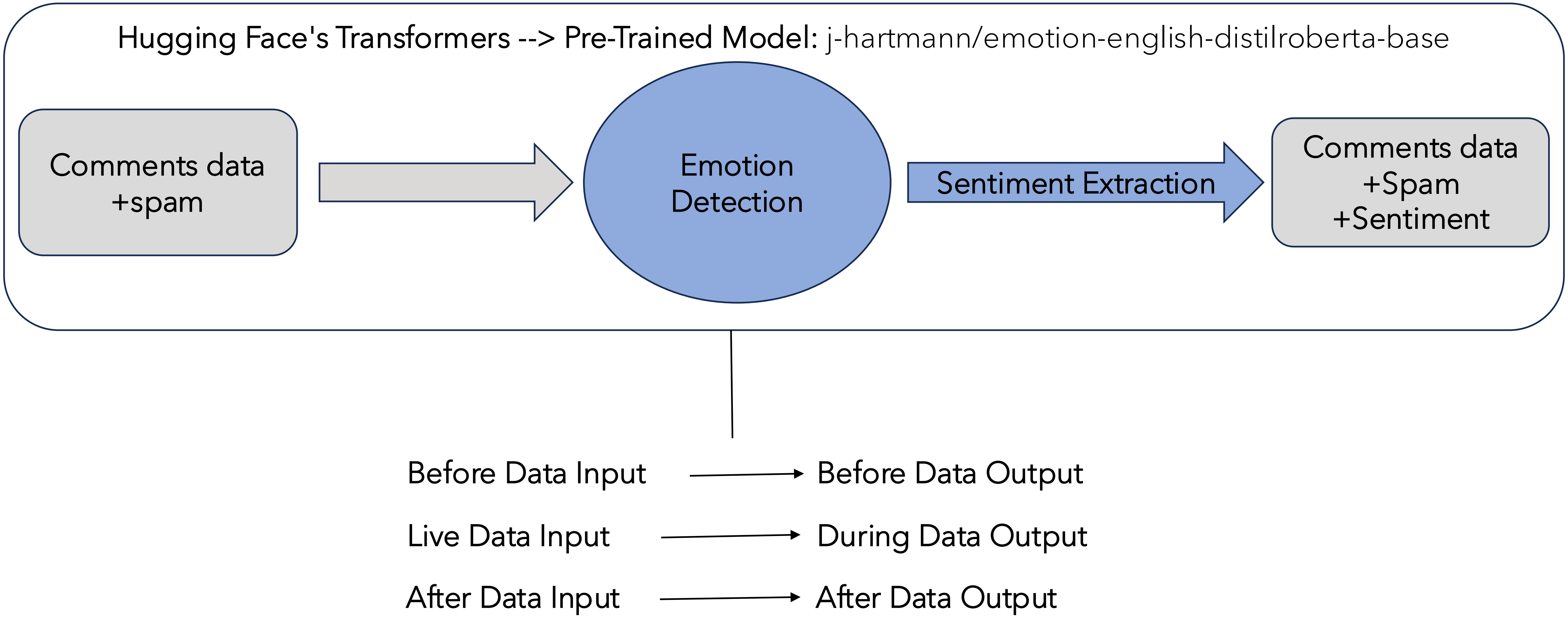
Spam Classifier – Predicting Spam Comments Results

RESEARCH QUESTION: What are the accounts that produce the most spam?

```python
# Identify accounts that produce the most spam
def find_most_spam_accounts(df, event_label):
    spam_by_account = df[df['is_spam'] == 1].groupby('account_id')['is_spam'].count().reset_index()
    spam_by_account = spam_by_account.sort_values(by='is_spam', ascending=False)
    print(f"\nTop accounts producing the most spam {event_label}:")
    print(spam_by_account.head())

# Find the top spam-producing accounts for each dataset
find_most_spam_accounts(before_spam_df, "before the event")
find_most_spam_accounts(live_spam_df, "during the event")
find_most_spam_accounts(after_spam_df, "after the event")
```

```
Top accounts producing the most spam before the event:
                    account_id  is_spam
209   UC2m4WXfD_7C-byzIGa5n9Rw       15
3001  UCuACr427uH-_WUNn4MEiQxw       13
2520  UClQmMFrzU4sQyGYQy2Ky7CQ       12
1003  UCJ-MHHBb6v5VNwSQo_YauyA       12
3280  UCzbvl9d8CDwUWmlI_8_4V6Q       12

Top accounts producing the most spam during the event:
                  account_id  is_spam
4   UCN20iPoQ5lNoPpIb-Zs5urw        3
15  UCoRCVQ6riUGZbTWczu4R1Nw        3
19  UCvyta-bVdS5C2p3sqr_4kjA        2
12  UCka-9LDcwDwZKEnjoKbZFVQ        2
3   UCEcNoMlftQZbxzHQjiDMvTw        2

Top accounts producing the most spam after the event:
                    account_id  is_spam
407   UC3vEFvlLGl2C7UGhzrYR1Nw       25
1472  UCIBgsuvMtnBRyvsSNfpR6Aw       23
682   UC7SoQLmV1iBAcNJv0RIMqog       21
3888  UClQmMFrzU4sQyGYQy2Ky7CQ       17
1605  UCJyt48GRehukCix-7dtmFgQ       16
```

Emotions Classifier – Structure



Emotions Classifier – Hugging Face's Transformer

```
def classify_emotion(comment, emotion_classifier):
    if isinstance(comment, str):
        try:
            result = emotion_classifier(comment, truncation=True, max_length=512) # 512 is the max length for the model's input
        except Exception as e:
            print(f"Error processing comment: {comment} - {e}")
            return None

        # Extract the label with the highest score
        return result if result else None
    return None
```

Use Hugging Face's Transformers to get emotions from text

```
def classify_emotions(input_filename, output_filename):
    df = pd.read_csv(input_filename, dtype={'content': str})

    # Load the emotion detection pipeline with a pre-trained model for emotion detection
    emotion_classifier = pipeline("text-classification", model="j-hartmann/emotion-english-distilroberta-base") # I choose this model because it's based on DistilBERT, which

    # Apply the classifier and create the emotion column
    df['emotion'] = df['content'].apply(lambda comment: classify_emotion(comment, emotion_classifier))

    # Save
    df.to_csv(output_filename, index=False)
```

```
input_filename_before = COMMENTS_BEFORE_WITH_SPAM_FILENAME
input_filename_live = COMMENTS_LIVE_WITH_SPAM_FILENAME
input_filename_after = COMMENTS_AFTER_WITH_SPAM_FILENAME
output_filename_before = COMMENTS_WITH_EMOTION_BEFORE_FILENAME
output_filename_live = COMMENTS_WITH_EMOTION_LIVE_FILENAME
output_filename_after = COMMENTS_WITH_EMOTION_AFTER_FILENAME


# Execute
classify_emotions(input_filename_before, output_filename_before)
classify_emotions(input_filename_live, output_filename_live)
classify_emotions(input_filename_after, output_filename_after)
```

Emotions Classifier – Emotion Classification Tuning / Extraction

```python
def separate_label_score(emotion_dic):
    if not pd.isna(emotion_dic):
        emotion_dic = str(emotion_dic).replace("'",'"')
        emotion_dic = json.loads(emotion_dic)

        label = emotion_dic[0]['label']
        score = emotion_dic[0]['score']

        if score <= 0.3: # Create neutral labels and scores
            label = 'neutral'
            score = 1-score

        return label, score
    else:
        return 'unknown', 0
```

Classified as 'neutral' low confidence scores

```python
def extract_emotion_data(input_filename, output_filename):
    df = pd.read_csv(input_filename)

    # Clean emotion column
    emotion_columns = df['emotion'].apply(separate_label_score)
    df[['emotion_label', 'emotion_score']] = pd.DataFrame(emotion_columns.tolist(), index=df.index)
    df = df.drop('emotion', axis=1) # Delete original column

    # Save
    df.to_csv(output_filename, index=False)
```

```python
input_filename_live = COMMENTS_WITH_EMOTION_LIVE_FILENAME
input_filename_before = COMMENTS_WITH_EMOTION_BEFORE_FILENAME
input_filename_after = COMMENTS_WITH_EMOTION_AFTER_FILENAME
output_filename_live = COMMENTS_WITH_EMOTIONS_CLEANED_LIVE_FILENAME
output_filename_before = COMMENTS_WITH_EMOTIONS_CLEANED_BEFORE_FILENAME
output_filename_after = COMMENTS_WITH_EMOTIONS_CLEANED_AFTER_FILENAME

# Execute
extract_emotion_data(input_filename_before, output_filename_before)
extract_emotion_data(input_filename_live, output_filename_live)
extract_emotion_data(input_filename_after, output_filename_after)
```

Data Analysis – Wordcloud

Most Frequent Words

Data Analysis – Extracting insights from wordcloud

RESEARCH QUESTION: What words are most associated with the topic? Can we identify iphone 16 features from the most used ones?

```python
# Visualizing top 100 topics
wordcloud = WordCloud(max_words=100).generate(text)
wordcloud.words_
```

```
{'iphone': 1.0,
 'apple': 0.950666186532229,
 'phone': 0.9503060857039971,
 'pro max': 0.7658144280398511,
 'iphone pro': 0.482415076221342,
 'one': 0.3754651302364662,
 'samsung': 0.3226503420957868,
 'bro': 0.3215700396110911,
 'thats': 0.3179690313287721,
 'video': 0.301884527667747,
 'think': 0.24846957148001442,
 'u': 0.23862681550834233,
 'even': 0.2381466810706998,
 'year': 0.23562597527307647,
 'camera': 0.234785740007202,
 'price': 0.23454567278838076,
 'thing': 0.21594046332973232,
 'better': 0.21594046332973232,
 'time': 0.2093386148121474,
 'make': 0.20873844676509423,
 'look': 0.20669787540511342,
 'design': 0.20633777457688152,
 'people': 0.20597767374864961,
 'lol': 0.20393710238866883,
```

Identified features

Still in use?

Data Analysis – Extract Emotion per Topic

Target interesting features

```
# Manually selecting topics for further analysis
# iphone takes sentiment from iphone pro and pro max too
'''INTERESTING_TOPICS = ['iphone', 'pro max', 'iphone pro', 'apple', 'android', 'samsung', 'camera', 'price', 'screen', 'battery', 'ultra', 'money', 'design', 'button', 'upg
'watch', 'galaxy', 'nokia', 'ipad', 'change', 'display', 'glass', 'protector', 'titanium', 'features', 'google', 'color', 'worth', 'update',  'xiaomi',
'buying', 'redmi', 'vision', 'expensive', 'support', 'intelligence', 'software', 'release', 'launch', 'quality', 'airpods', 'games', 'innovation',
'storage'
]
'''
INTERESTING_FEATURES=['camera', 'price', 'money', 'screen', 'battery', 'design', 'case', 'display', 'titanium', 'color', 'software', 'quality', 'innovation', 'storage']
```

4 - Extract emotions per topic

```
def extract_emotion_per_topic(input_filename, output_filename):
    df = pd.read_csv(input_filename)
    topic_emotions = {}

    for topic in INTERESTING_FEATURES:
        filtered_df = df[df['content'].str.contains(topic, case=False, na=False)]
        emotion_counts = {}
        emotions = {}
        count = 0

        for _, row in filtered_df.iterrows():
            count += 1

            if row['emotion_label'] in emotions:
                emotions[row['emotion_label']] += 1
            else:
                emotions[row['emotion_label']] = 1

        for emotion_key in emotions.keys():
            emotions[emotion_key] = emotions[emotion_key]/count

        topic_emotions[topic] = emotions

    with open(output_filename, 'w') as f:
        json.dump(topic_emotions, f)
```

Data Analysis – Convert Emotions to Sentiment

5 - Analyze sentiment

```
def convert_emotion_to_sentiment_label(emotion):
    positive_emotions = ['joy']
    neutral_emotions = ['neutral', 'surprise']
    negative_emotions = ['anger', 'disgust', 'fear', 'sadness']

    if emotion in positive_emotions:
        return 'positive'
    elif emotion in neutral_emotions:
        return 'neutral'
    elif emotion in negative_emotions:
        return 'negative'

def convert_emotions_to_sentiment_df(input_filename, output_filename):
    df = pd.read_csv(input_filename)
    df = df.rename(columns={'emotion_label':'sentiment', 'emotion_score':'sentiment_score'})
    df['sentiment'] = df['sentiment'].apply(convert_emotion_to_sentiment_label)

    df.to_csv(output_filename)
```

Set "surprise" as "neutral" because it can be both positive or negative

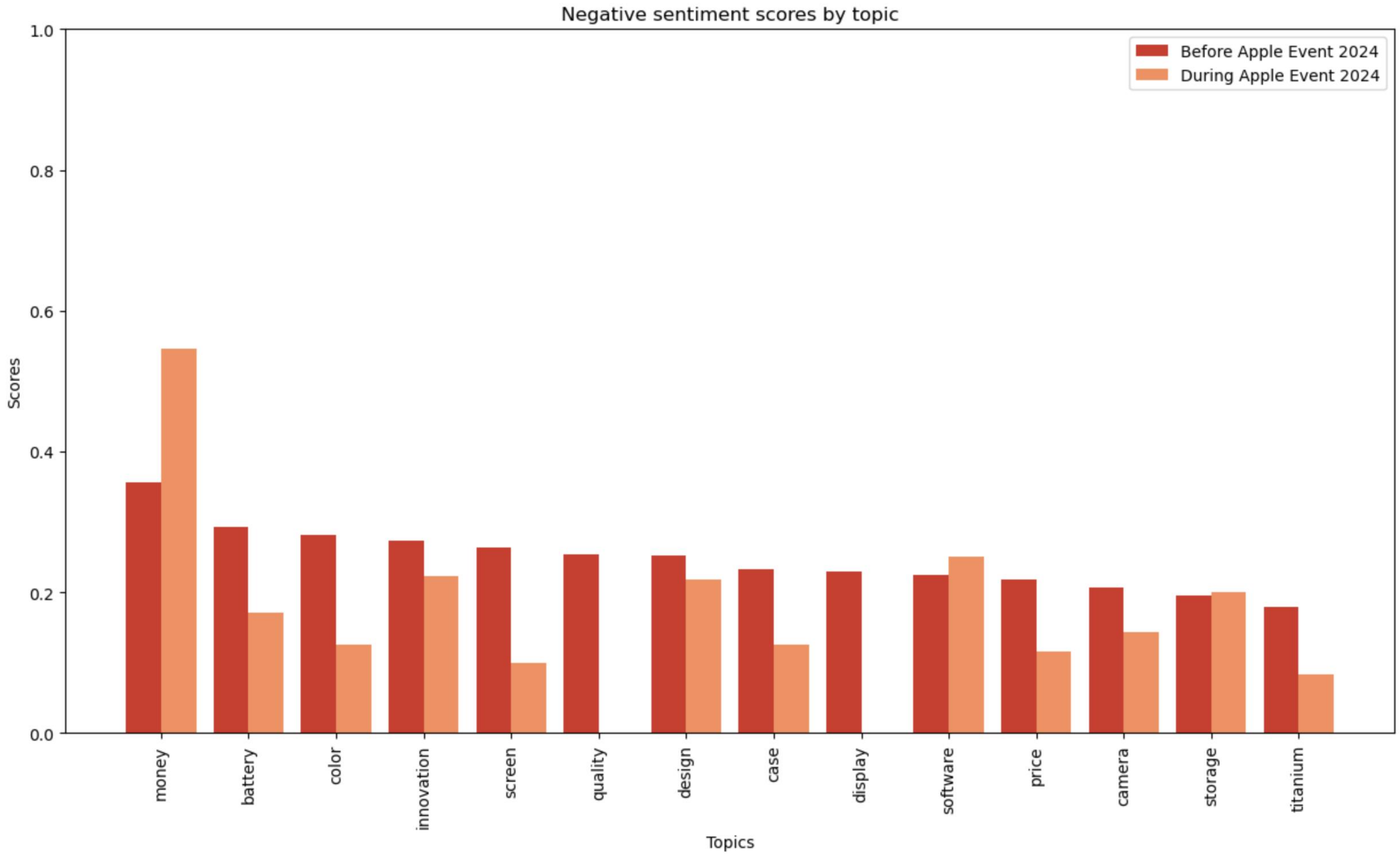
```
def convert_emotions_to_sentiment(input_filename, output_filename):
    with open(input_filename, 'r') as f:
        topic_emotions = json.load(f)

    positive_emotions = ['joy']
    neutral_emotions = ['neutral', 'surprise']
    negative_emotions = ['anger', 'disgust', 'fear', 'sadness']
    topics_sentiment = {}

    for topic_key in topic_emotions.keys():
        topic_sentiment = {'positive':0, 'neutral':0, 'negative':0}

        for emotion_key in topic_emotions[topic_key].keys():
            if emotion_key in positive_emotions:
                topic_sentiment['positive'] += topic_emotions[topic_key][emotion_key]
            elif emotion_key in neutral_emotions:
                topic_sentiment['neutral'] += topic_emotions[topic_key][emotion_key]
            elif emotion_key in negative_emotions:
                topic_sentiment['negative'] += topic_emotions[topic_key][emotion_key]

        topics_sentiment[topic_key] = topic_sentiment

    with open(output_filename, 'w') as f:
        json.dump(topics_sentiment, f)
```

Data Analysis – Sentiment per topic

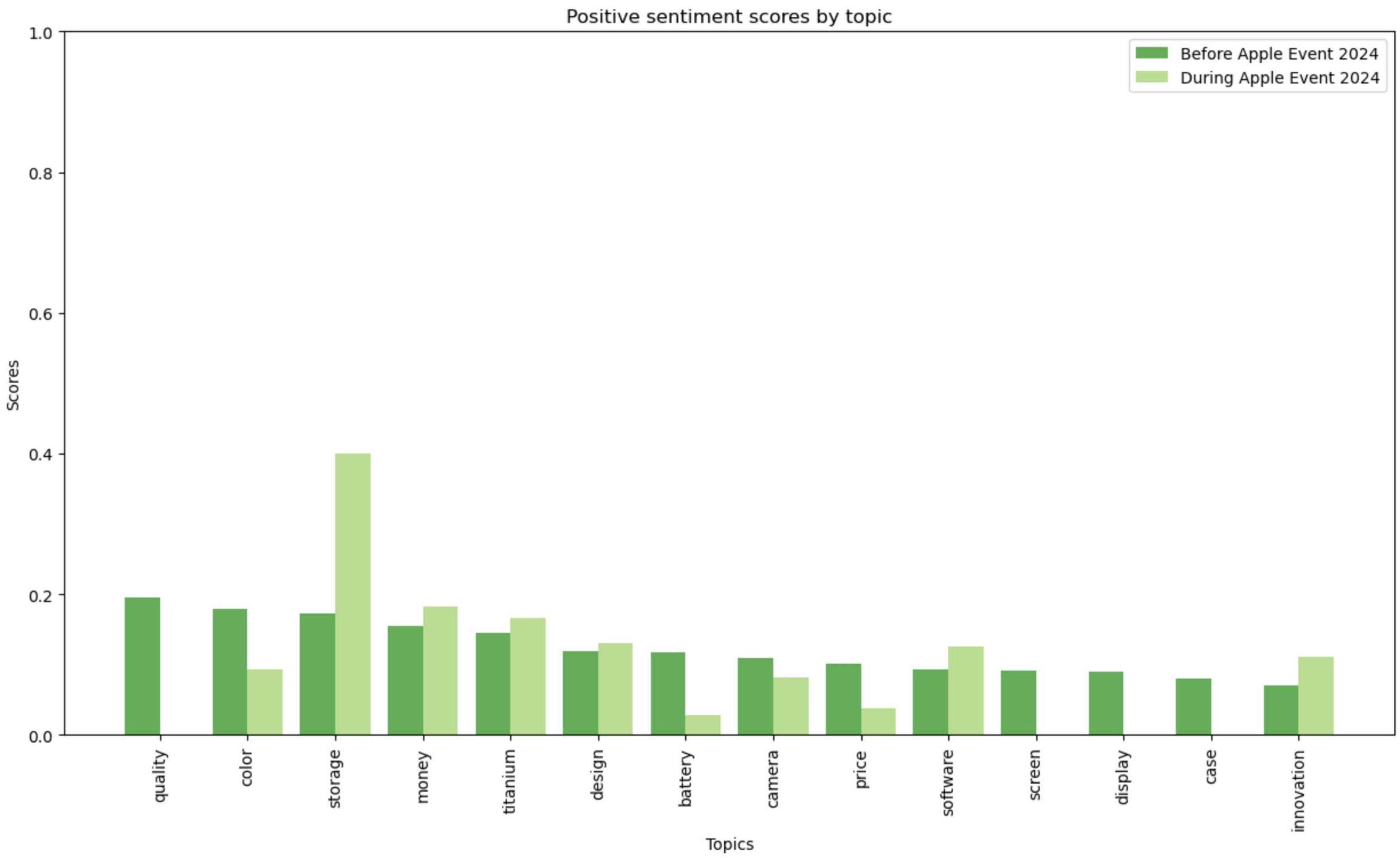
RESEARCH QUESTION: What sentiment is associated with iphone 16 features?

RESEARCH QUESTION: Have feature sentiments shifted before, during and after the apple event?

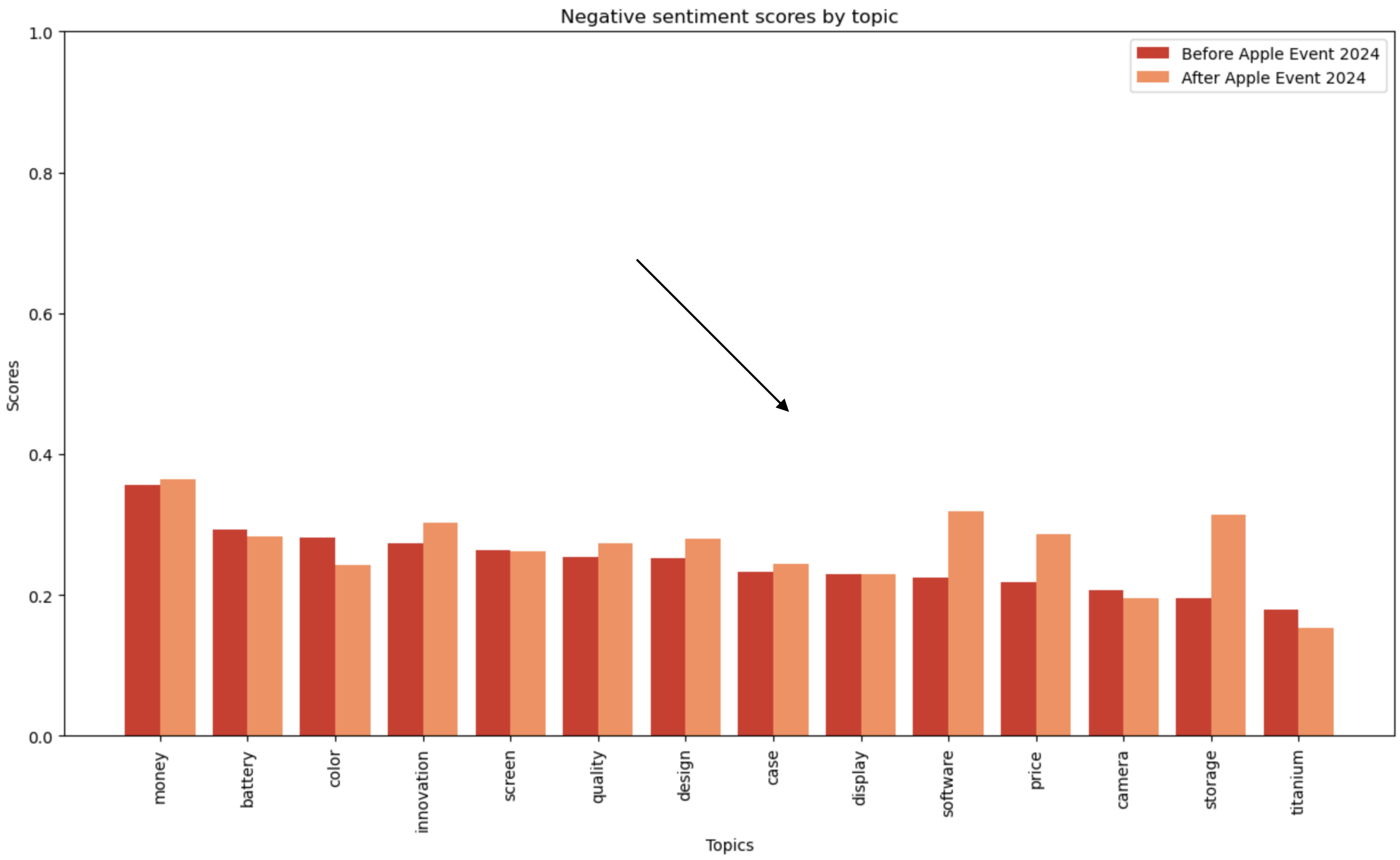
Data Analysis – Results



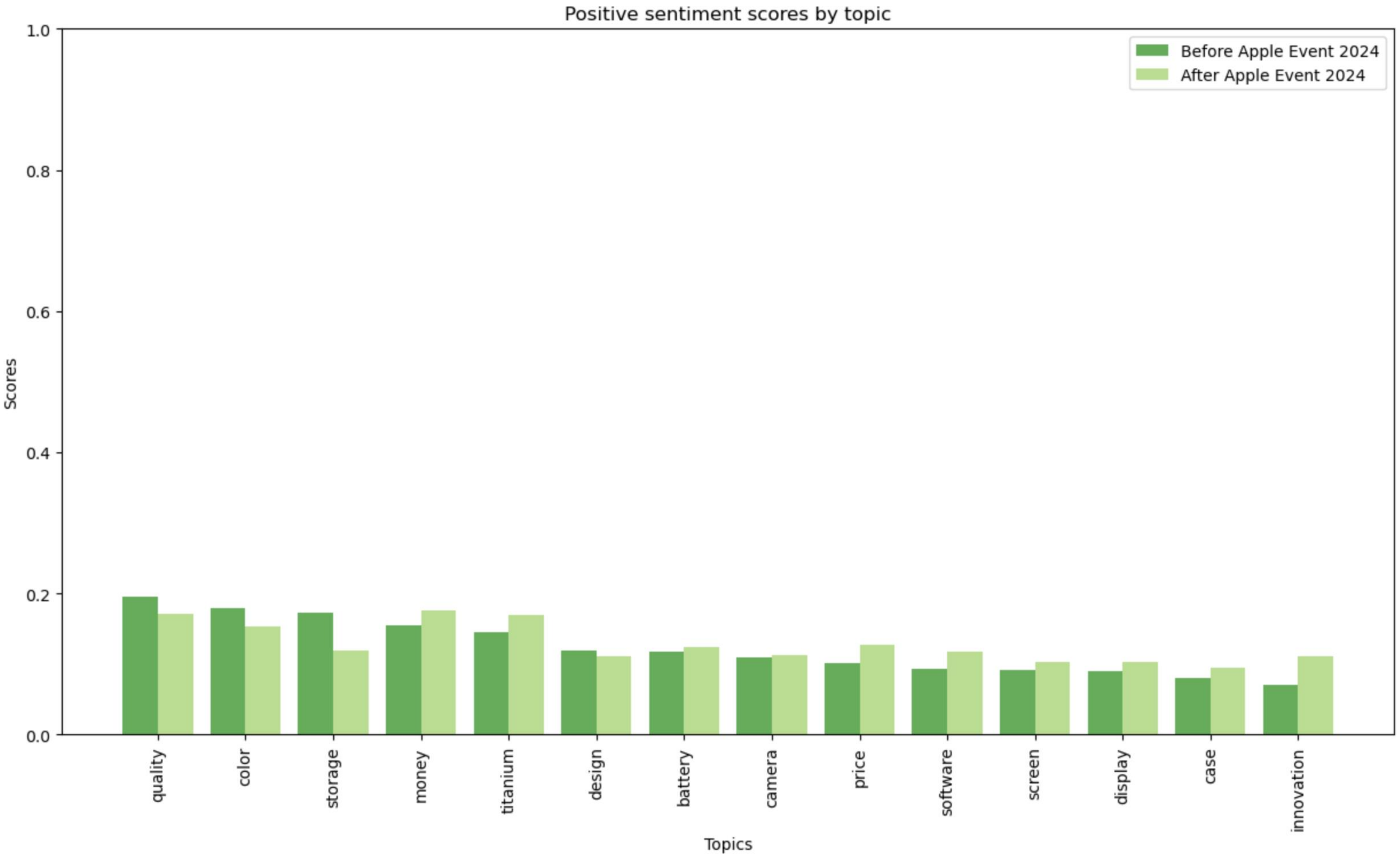
Data Analysis – Results



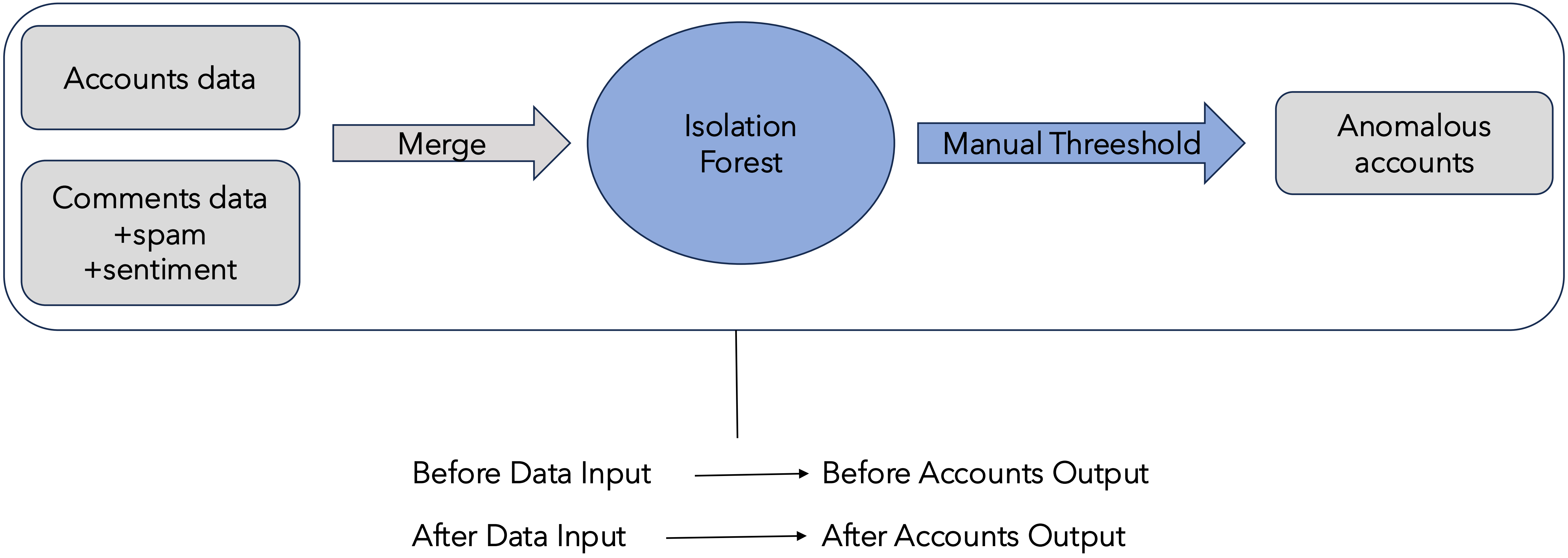
Data Analysis – Results



Data Analysis – Results



Anomalies Classifier – Structure



Anomalies Classifier – Cleaning and Merging

```
Account data:
<class 'pandas.core.frame.DataFrame'>
Index: 190397 entries, 0 to 190455
Data columns (total 8 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   account_id              190397 non-null  object
 1   title                   190397 non-null  object
 2   account_name            190397 non-null  object
 3   published_at            190397 non-null  object
 4   view_count              190397 non-null  int64
 5   subscriber_count        190397 non-null  float64
 6   hidden_subscriber_count 190397 non-null  bool
 7   video_count             190397 non-null  int64
dtypes: bool(1), float64(1), int64(2), object(4)
memory usage: 11.8+ MB
None

Comment data:
<class 'pandas.core.frame.DataFrame'>
Index: 131079 entries, 0 to 131380
Data columns (total 17 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   Unnamed: 0             131079 non-null  int64
 1   content                131079 non-null  object
 2   text_length            131079 non-null  int64
 3   num_links              131079 non-null  int64
 4   num_special_chars      131079 non-null  int64
 5   capitalization_ratio   131079 non-null  float64
 6   num_digits             131079 non-null  int64
 7   num_repeated_chars     131079 non-null  int64
 8   sentiment_score        131079 non-null  float64
 9   is_spam                131079 non-null  int64
 10  account_id             131079 non-null  object
 11  comment_id             131079 non-null  object
 12  video_id               131079 non-null  object
 13  original_comment_text  131079 non-null  object
 14  like_count             131079 non-null  float64
 15  sentiment              131079 non-null  object
 16  sentiment_score        131079 non-null  float64
dtypes: float64(4), int64(7), object(6)
memory usage: 18.0+ MB
None
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 131035 entries, 0 to 131078
Data columns (total 24 columns):
 #   Column                   Non-Null Count   Dtype
---  ------                   --------------   -----
 0   Unnamed: 0               131035 non-null  int64
 1   content                  131035 non-null  object
 2   text_length              131035 non-null  int64
 3   num_links                131035 non-null  int64
 4   num_special_chars        131035 non-null  int64
 5   capitalization_ratio     131035 non-null  float64
 6   num_digits               131035 non-null  int64
 7   num_repeated_chars       131035 non-null  int64
 8   sentiment_score          131035 non-null  float64
 9   is_spam                  131035 non-null  int64
 10  account_id               131035 non-null  object
 11  comment_id               131035 non-null  object
 12  video_id                 131035 non-null  object
 13  original_comment_text    131035 non-null  object
 14  like_count               131035 non-null  float64
 15  sentiment                131035 non-null  object
 16  sentiment_score          131035 non-null  float64
 17  title                    131035 non-null  object
 18  account_name             131035 non-null  object
 19  published_at             131035 non-null  datetime64[ns]
 20  view_count               131035 non-null  float64
 21  subscriber_count         131035 non-null  float64
 22  hidden_subscriber_count  131035 non-null  object
 23  video_count              131035 non-null  float64
dtypes: datetime64[ns](1), float64(7), int64(7), object(9)
memory usage: 25.0+ MB
```

Anomalies Classifier – Feature Extraction

```python
# Extract features from the accounts dataset
features_X = pd.DataFrame()
features_X['account_id'] = merged_df['account_id']
features_X['video_id'] = merged_df['video_id']

# Convert relevant columns to numeric
features_X['video_count'] = pd.to_numeric(merged_df['video_count'], errors='coerce').fillna(0)
features_X['view_count'] = pd.to_numeric(merged_df['view_count'], errors='coerce').fillna(0)
features_X['subscriber_count'] = pd.to_numeric(merged_df['subscriber_count'], errors='coerce').fillna(0)

merged_df['published_at'] = pd.to_datetime(merged_df['published_at'], errors='coerce') # Ensure 'published_at' is in datetime format
features_X['account_age_days'] = (datetime.now() - merged_df['published_at']).dt.days
features_X['upload_frequency'] = features_X['video_count'] / (features_X['account_age_days'] + 1)
features_X['view_to_subscriber_ratio'] = features_X['view_count'] / (features_X['subscriber_count'] + 1)
features_X['subscriber_to_video_ratio'] = features_X['subscriber_count'] / (features_X['video_count'] + 1)
features_X['view_per_video'] = features_X['view_count'] / (features_X['video_count'] + 1)
features_X['title_length'] = merged_df['title'].apply(lambda x: len(str(x)))
#features_X['description_length'] = merged_df['description'].apply(lambda x: len(str(x)))

# Extract features from the comments dataset
# Calculate comment count per account
comment_counts = merged_df.groupby('account_id').size().reset_index(name='comment_count')

# Calculate comment length for each comment
merged_df['comment_length'] = merged_df['original_comment_text'].apply(lambda x: len(str(x)))

# Calculate average comment length per account
avg_comment_length = merged_df.groupby('account_id')['comment_length'].mean().reset_index(name='avg_comment_length')

# Calculate like-to-comment ratio per account
like_comment_ratio = merged_df.groupby('account_id')['like_count'].sum() / comment_counts.set_index('account_id')['comment_count']
like_comment_ratio = like_comment_ratio.reset_index(name='like_to_comment_ratio')

# Calculate spam count per account
spam_count = merged_df.groupby('account_id')['is_spam'].sum().reset_index(name='spam_count')

# Calculate sentiment count (positive/neutral/negative) per account
sentiment_count = merged_df.groupby(['account_id', 'sentiment']).size().unstack(fill_value=0).reset_index() # Unstack to separate resulted column (since groupby is by 2 valu
sentiment_count.columns = ['account_id', 'negative_sentiment_count', 'neutral_sentiment_count', 'positive_sentiment_count'] # Rename resulted column

# Merge features
features_X = features_X.merge(comment_counts, on='account_id', how='left')
features_X = features_X.merge(avg_comment_length, on='account_id', how='left')
features_X = features_X.merge(like_comment_ratio, on='account_id', how='left')
features_X = features_X.merge(spam_count, on='account_id', how='left')
features_X = features_X.merge(sentiment_count, on='account_id', how='left')

# Calculate spam ratio per account
features_X['spam_ratio'] = features_X['spam_count'] / features_X['comment_count']
```

```python
features_X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 95847 entries, 0 to 131034
Data columns (total 19 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   account_id                 95847 non-null  object
 1   video_id                   95847 non-null  object
 2   video_count                95847 non-null  float64
 3   view_count                 95847 non-null  float64
 4   subscriber_count           95847 non-null  float64
 5   account_age_days           95847 non-null  int64
 6   upload_frequency           95847 non-null  float64
 7   view_to_subscriber_ratio   95847 non-null  float64
 8   subscriber_to_video_ratio  95847 non-null  float64
 9   view_per_video             95847 non-null  float64
 10  title_length               95847 non-null  int64
 11  comment_count              95847 non-null  int64
 12  avg_comment_length         95847 non-null  float64
 13  like_to_comment_ratio      95847 non-null  float64
 14  spam_count                 95847 non-null  int64
 15  negative_sentiment_count   95847 non-null  int64
 16  neutral_sentiment_count    95847 non-null  int64
 17  positive_sentiment_count   95847 non-null  int64
 18  spam_ratio                 95847 non-null  float64
dtypes: float64(10), int64(7), object(2)
memory usage: 14.6+ MB
```

Anomalies Classifier – Pipelines

3 - Prepare pipelines

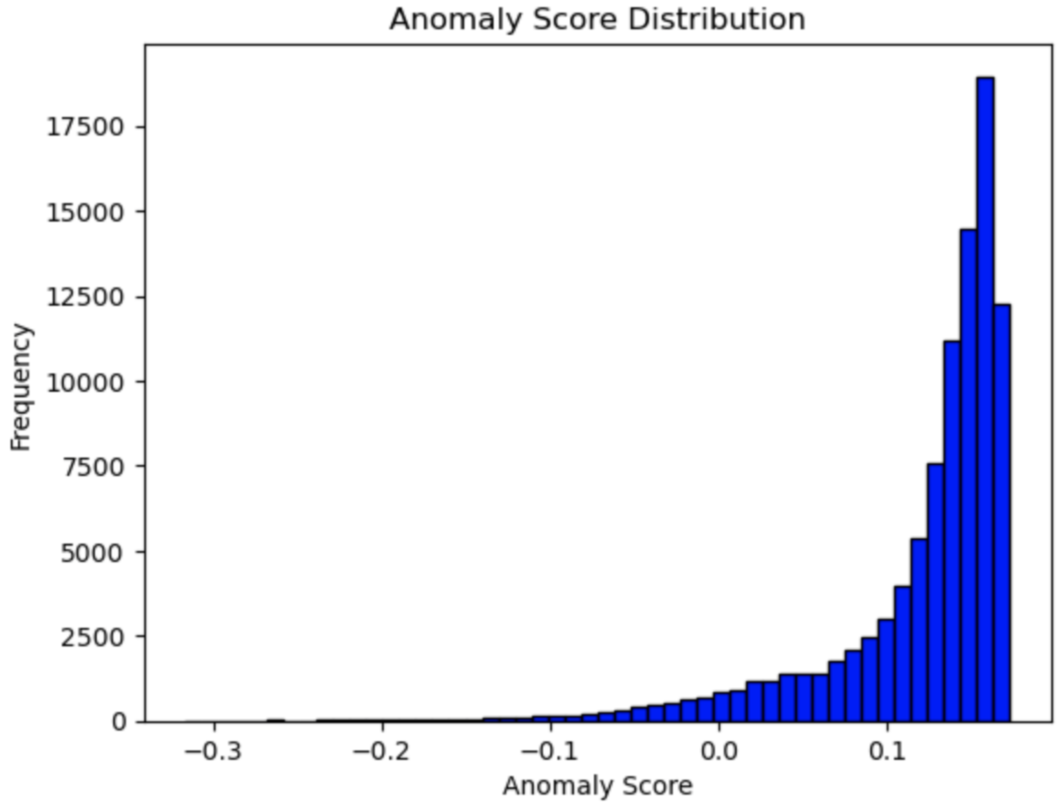
```
# Model Pipeline
features_conversion = ColumnTransformer([
    ('scaler', StandardScaler(), ['account_age_days', 'upload_frequency', 'view_to_subscriber_ratio', 'subscriber_to_video_ratio', 'view_per_video', 'title_length', 'comment
    ]
)

model_pipeline = Pipeline([
    ('features_conversion', features_conversion),
    ('classifier', IsolationForest(n_estimators=100, contamination=0.05, random_state=42))
])
```

Anomalies Classifier – Anomaly Detection Phase

```
# Prepare final df
final_df = features_X[['account_id', 'video_id', 'spam_count', 'negative_sentiment_count', 'neutral_sentiment_count', 'positive_sentiment_count']]

# Prepare feature matrix
X = features_X[['account_age_days', 'upload_frequency', 'view_to_subscriber_ratio', 'subscriber_to_video_ratio', 'view_per_video', 'title_length', 'comment_count', 'avg_comm

# Fit the model
model_pipeline.fit(X)
anomaly_scores = model_pipeline.decision_function(X)
```

Anomalies Classifier – "Hyperparameter Tuning"

```python
# Plot the distribution of anomaly scores
plt.hist(anomaly_scores, bins=50, color='blue', edgecolor='black')
plt.title('Anomaly Score Distribution')
plt.xlabel('Anomaly Score')
plt.ylabel('Frequency')
plt.show()
```




```python
threshold = -0.1
```

Anomalies Classifier – Assign Prediction (and output file)

```
threshold = -0.1

# Classify points as anomalous if their score is below the threshold
final_df = final_df.copy() # Avoid using a view of the dataset before loc
final_df.loc[:, 'is_anomalous'] = anomaly_scores < threshold
```

```
final_df.head()
```

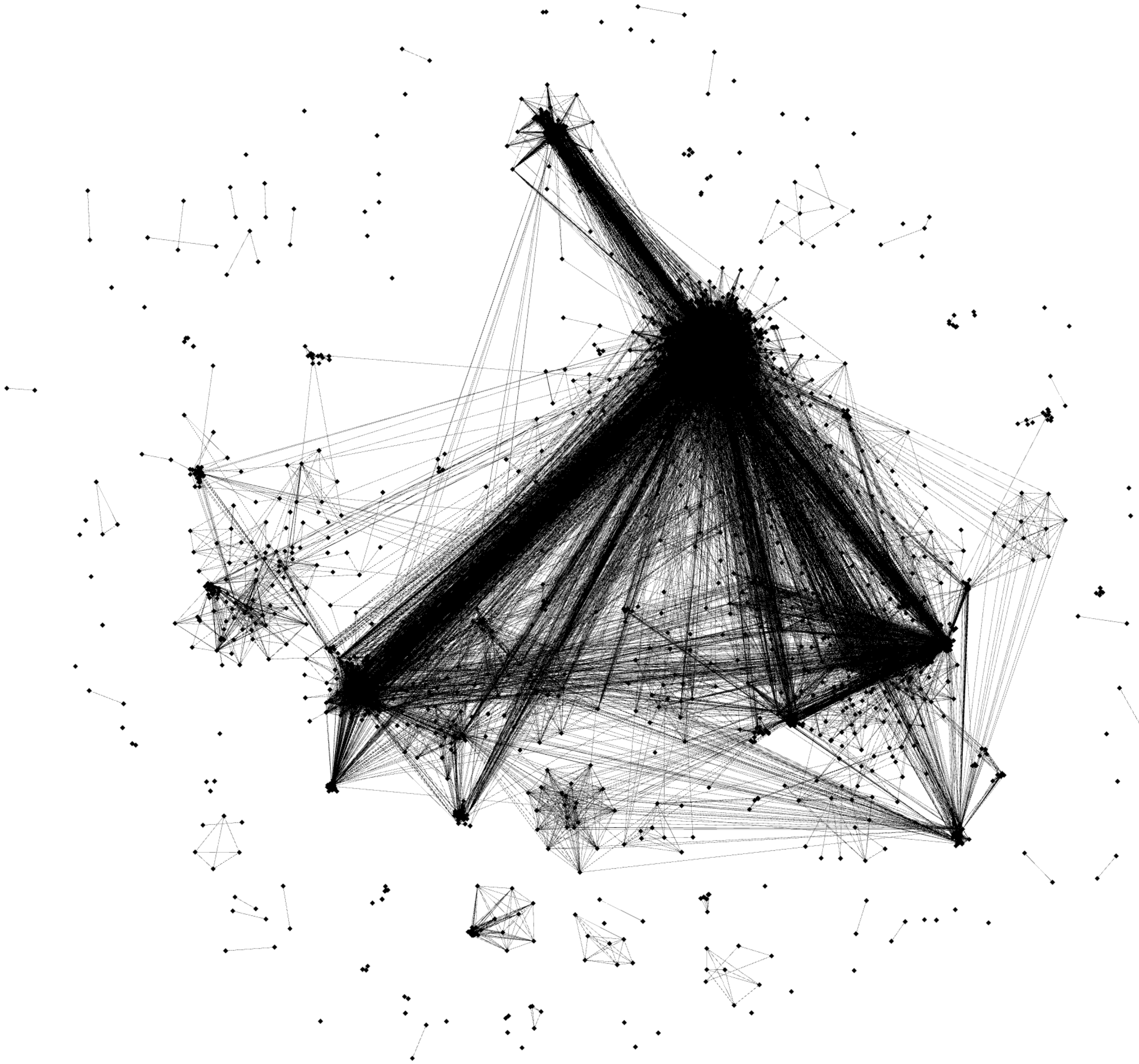
	account_id	video_id	spam_count	negative_sentiment_count	neutral_sentiment_count	positive_sentiment_count	is_anomalous
0	UCgEAEygar_JKymja6lmZpSw	h3BKjZMGolw	0	0	0	1	False
1	UCszWY8pgNZASj7qaQY2b5UQ	h3BKjZMGolw	0	1	0	0	False
2	UClmAcrwg6ddcpD3N7scfqSQ	h3BKjZMGolw	0	1	0	0	False
3	UCxcf_ul15ynwQsf7xjU34uw	h3BKjZMGolw	0	0	0	1	False
4	UCYRdLNCjFReh61plmbPoqCg	h3BKjZMGolw	0	1	0	0	False

```
final_df.to_csv(ANOMALOUS_ACCOUNTS_AFTER_FILENAME, index=False)
```

Network Science – Before

order: 4190

size: 55778

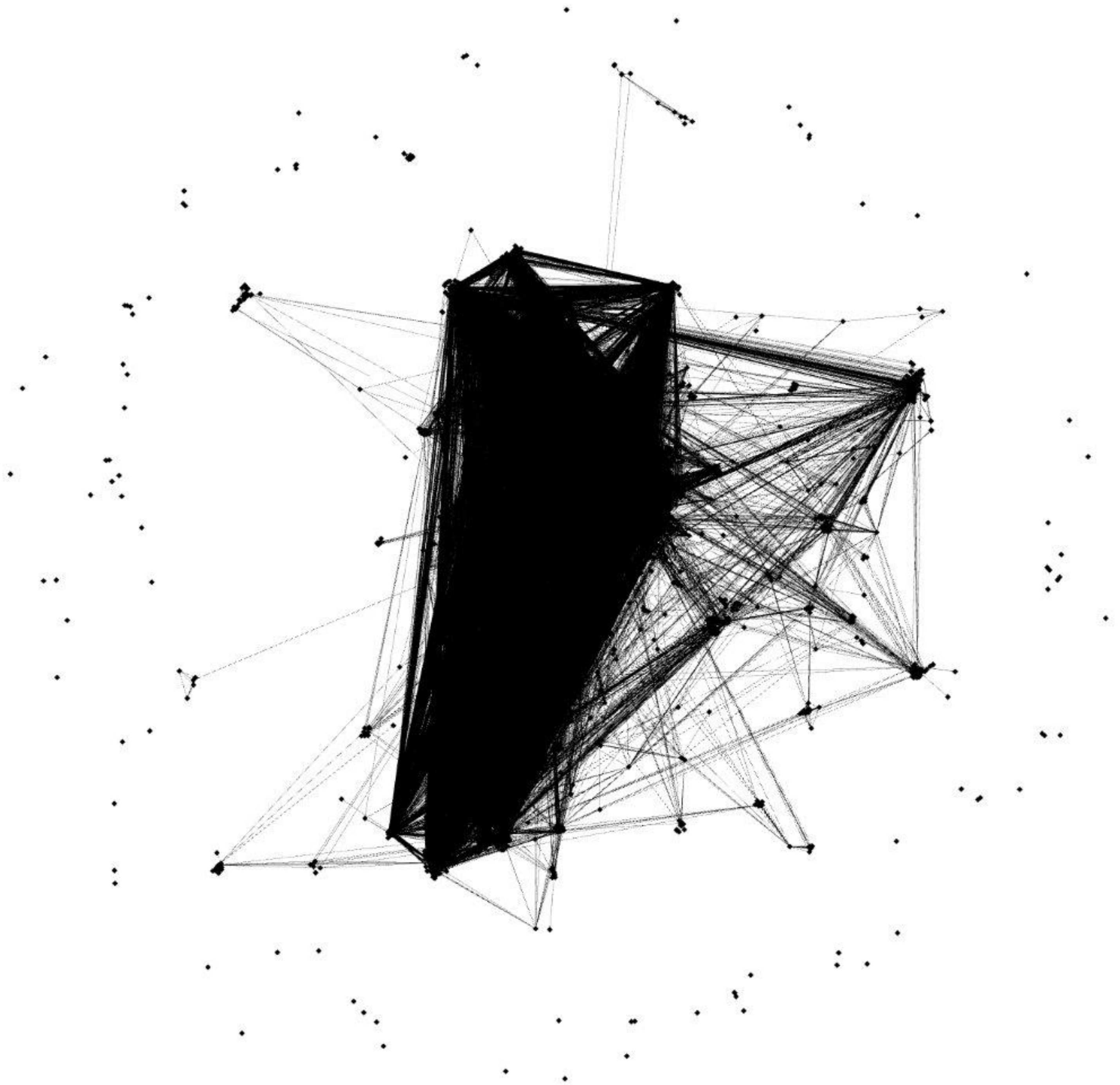


Network Science – After

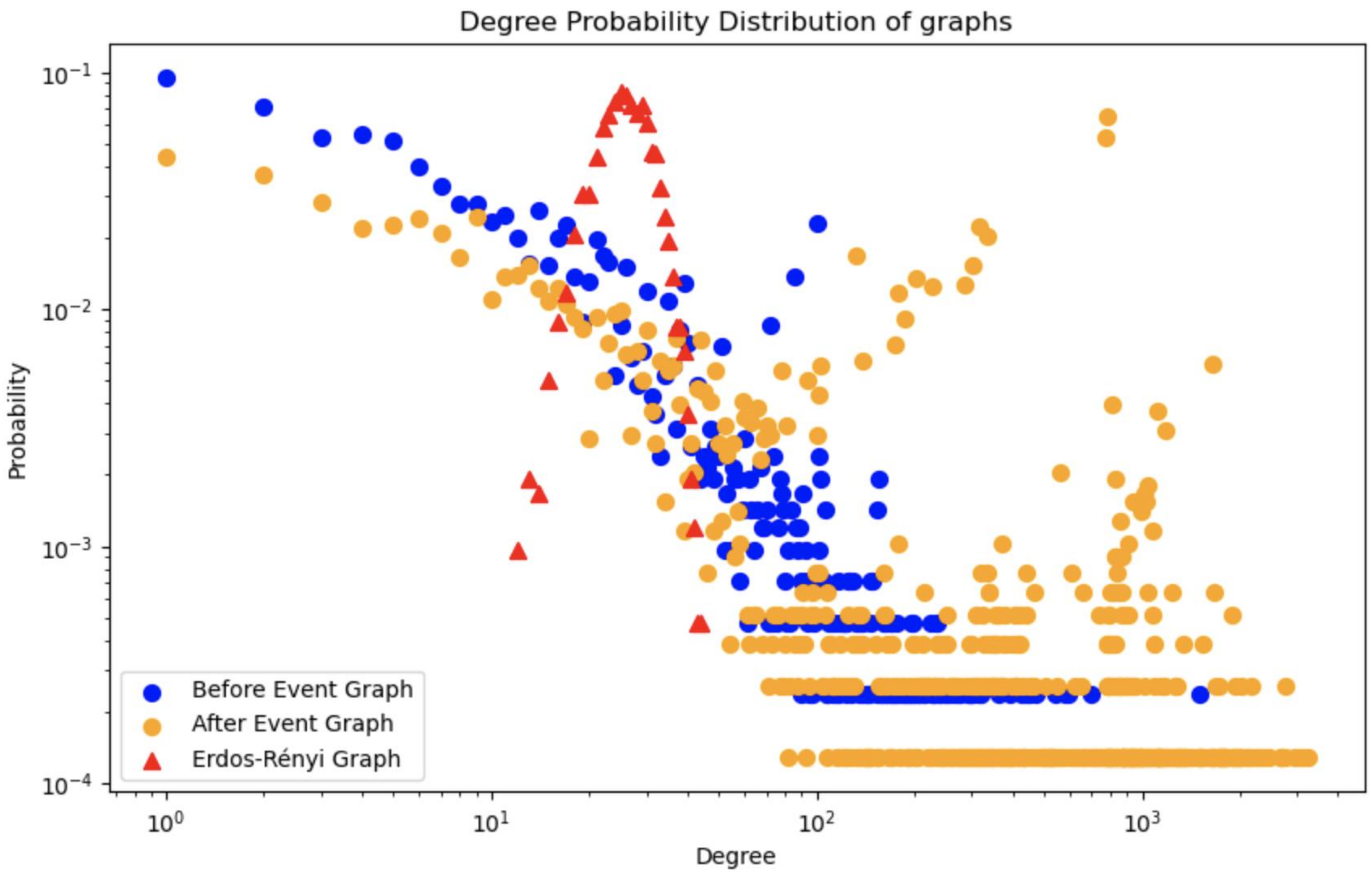
order: 7789

size: 989232

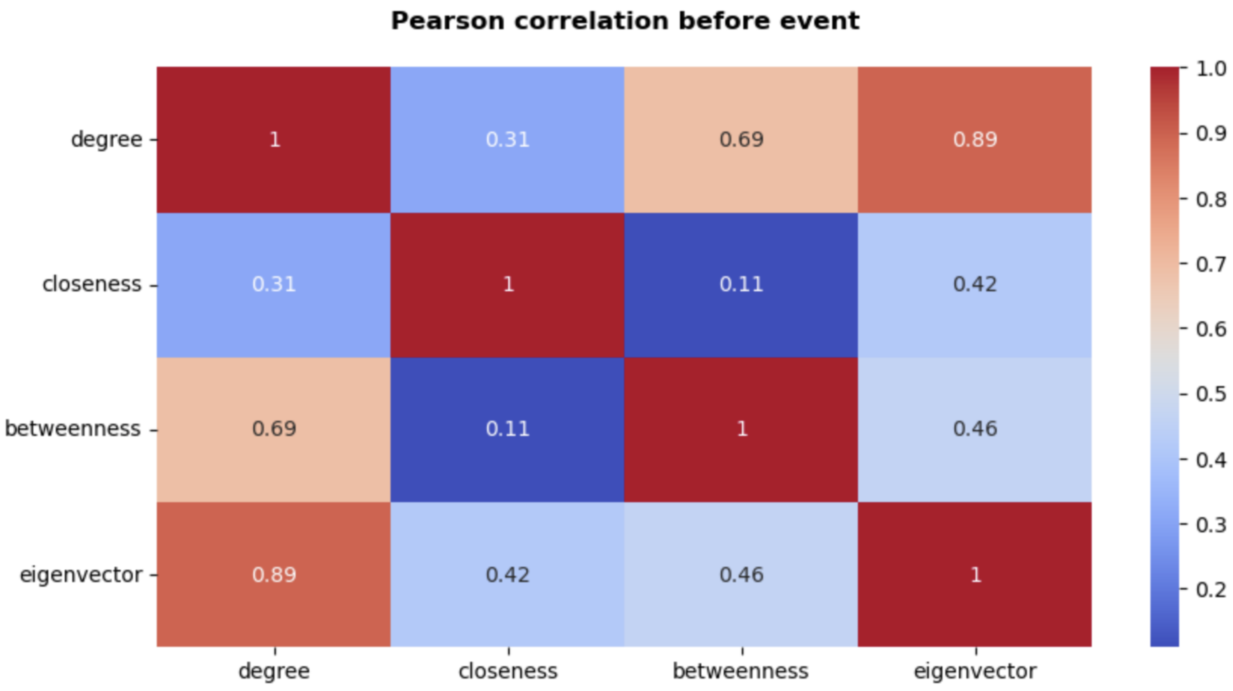
There's an evident increase in engagement in the graph obtained after the event, the size increases exponentially as more and more people comment on the same videos

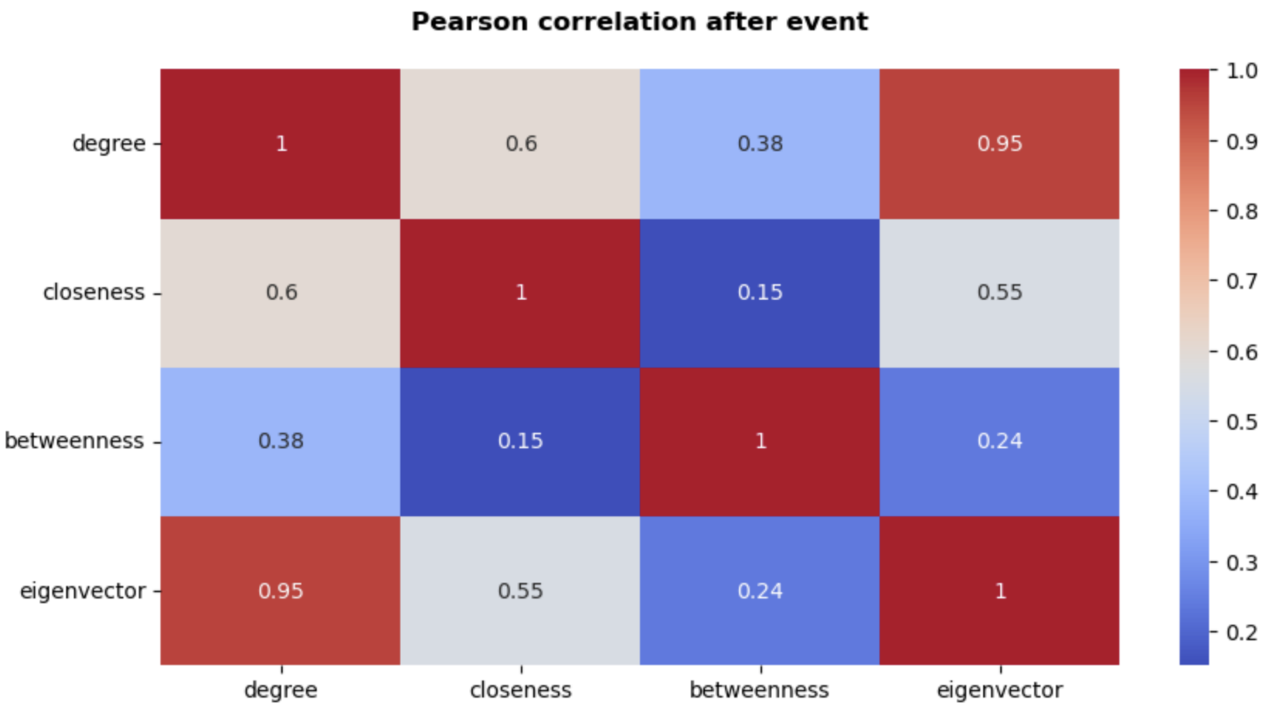


Network Science – Degree Probability Distribution



Network Science – Centrality (Pearson Correlation)





Network Science – Centrality

```
Degree Centrality top nodes before the event: [('UCtLU0qhHo4J3tsy1HrltLkQ', 0.359), ('UCDy4o0IH0dEp0lk23Bhetzg', 0.1666), ('UCDVUh8E7QVDBs6wNWKewSfw', 0.1418), ('UCalY2ddRg
xnObnsgFh9Ms2g', 0.138), ('UCoVQZdNBZW0dbdyf6ly1QmA', 0.1299)]

Degree Centrality top nodes after the event: [('UCLtRxV85jYvJVm-XaFths7A', 0.4138), ('UCWBgism_fFLIuN52ggvePgA', 0.4023), ('UCQmZ-wOK2fJy1Ojn2Ou6sVg', 0.3905), ('UCiex9LaxK
UBrwLotY50ngSQ', 0.3811), ('UC5vz19CfEaZxDjeKXa4ySsw', 0.3801)]
```

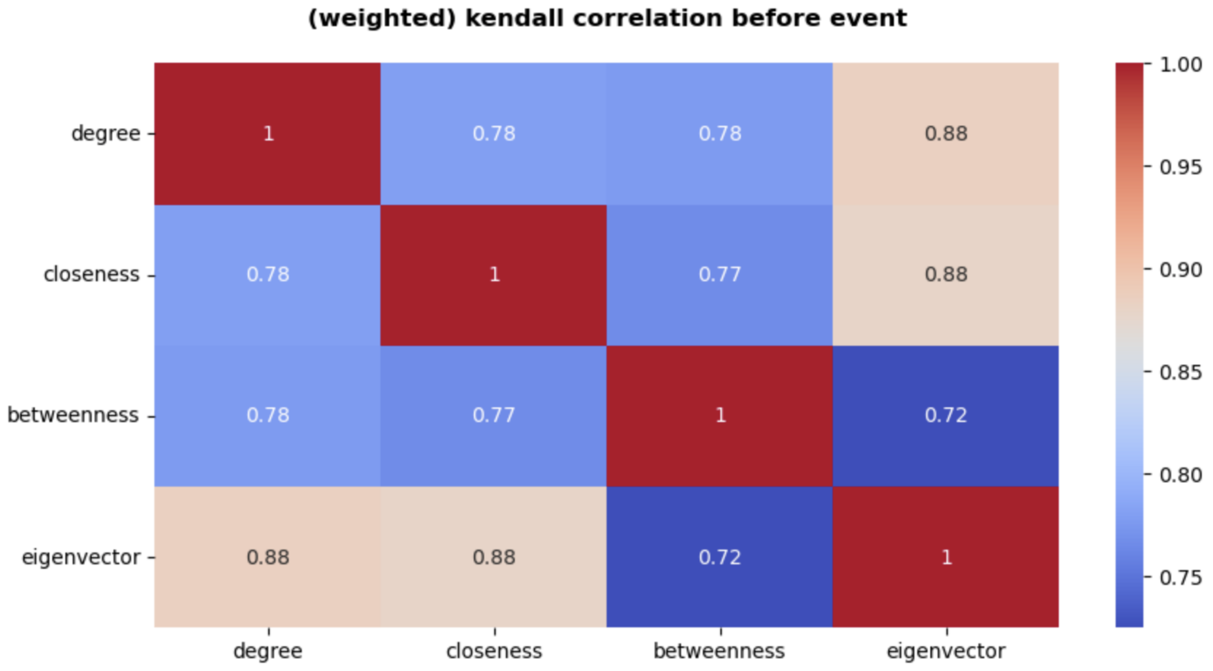
```
Closeness Centrality top nodes before the event: [('UCtLU0qhHo4J3tsy1HrltLkQ', 0.5275322920764226), ('UCDVUh8E7QVDBs6wNWKewSfw', 0.4472912674086531), ('UCDy4o0IH0dEp0lk23Bh
etzg', 0.4459611979726715), ('UCoVQZdNBZW0dbdyf6ly1QmA', 0.4449688254218607), ('UCalY2ddRgxnObnsgFh9Ms2g', 0.4330830019055988)]

Closeness Centrality top nodes after the event: [('UCLtRxV85jYvJVm-XaFths7A', 0.588748320463633), ('UCWBgism_fFLIuN52ggvePgA', 0.5813450092977903), ('UCQmZ-wOK2fJy1Ojn2Ou6s
Vg', 0.5748394387865312), ('UCiex9LaxKUBrwLotY50ngSQ', 0.5729249398423889), ('UC5vz19CfEaZxDjeKXa4ySsw', 0.5717735099563334)]
```

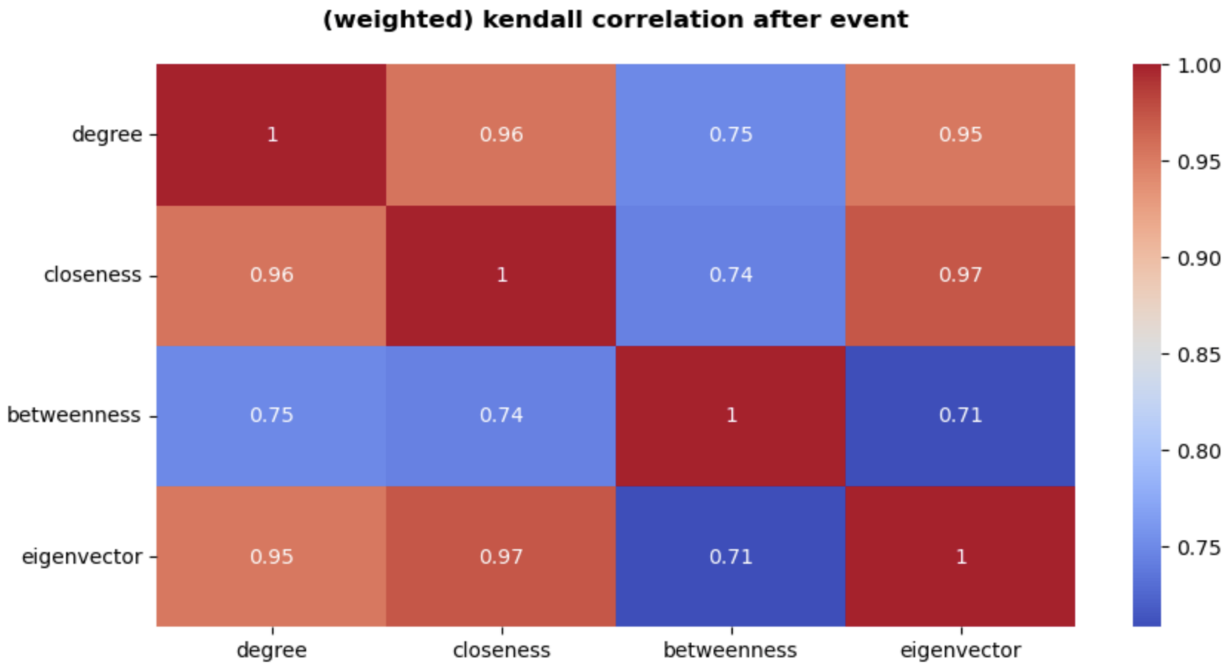
```
Betweenness Centrality top nodes before the event: [('UCtLU0qhHo4J3tsy1HrltLkQ', 0.2618187993486857), ('UCDy4o0IH0dEp0lk23Bhetzg', 0.0610060398418432), ('UCQgRXb3-WwryxUX9d
U0l1Ww', 0.04718718358793657), ('UCoVQZdNBZW0dbdyf6ly1QmA', 0.043887397119769925), ('UCDVUh8E7QVDBs6wNWKewSfw', 0.04055693889124225)]

Betweenness Centrality top nodes after the event: [('UCiex9LaxKUBrwLotY50ngSQ', 0.057438524723535674), ('UCWBgism_fFLIuN52ggvePgA', 0.0495113603756372 7), ('UCLtRxV85jYvJVm-
XaFths7A', 0.04288429765788693), ('UCJwjWjde_l7amecXUpDW00g', 0.04171723767324198), ('UC1DfJEKqI1lsxmI_3SYCGMw', 0.024692728704489162)]
```

```
Eigenvector Centrality top nodes before the event: [('UCtLU0qhHo4J3tsy1HrltLkQ', 0.22858210616225166), ('UCalY2ddRgxnObnsgFh9Ms2g', 0.1363521831182538), ('UCDy4o0IH0dEp0lk2
3Bhetzg', 0.13573457297049912), ('UCoVQZdNBZW0dbdyf6ly1QmA', 0.12427504230438396), ('UC7tDR1svtHPvxv3aiBAdr4g', 0.12056586657441391)]

Eigenvector Centrality top nodes after the event: [('UCQmZ-wOK2fJy1Ojn2Ou6sVg', 0.0480737178670853), ('UCLtRxV85jYvJVm-XaFths7A', 0.04789861123218195 5), ('UCWBgism_fFLIuN52
ggvePgA', 0.047685271643825355), ('UC5vz19CfEaZxDjeKXa4ySsw', 0.04764677393080773), ('UCBpHHVuYafz6Ve-5Xp0oodA', 0.04737543140084343)]
```

The values differs between different centrality measures but the rankings looks similar. There are accounts that appear in the top 5 of every centrality measure

Network Science – Centrality (Kendall Correlation)





Network Science – Spam in central nodes

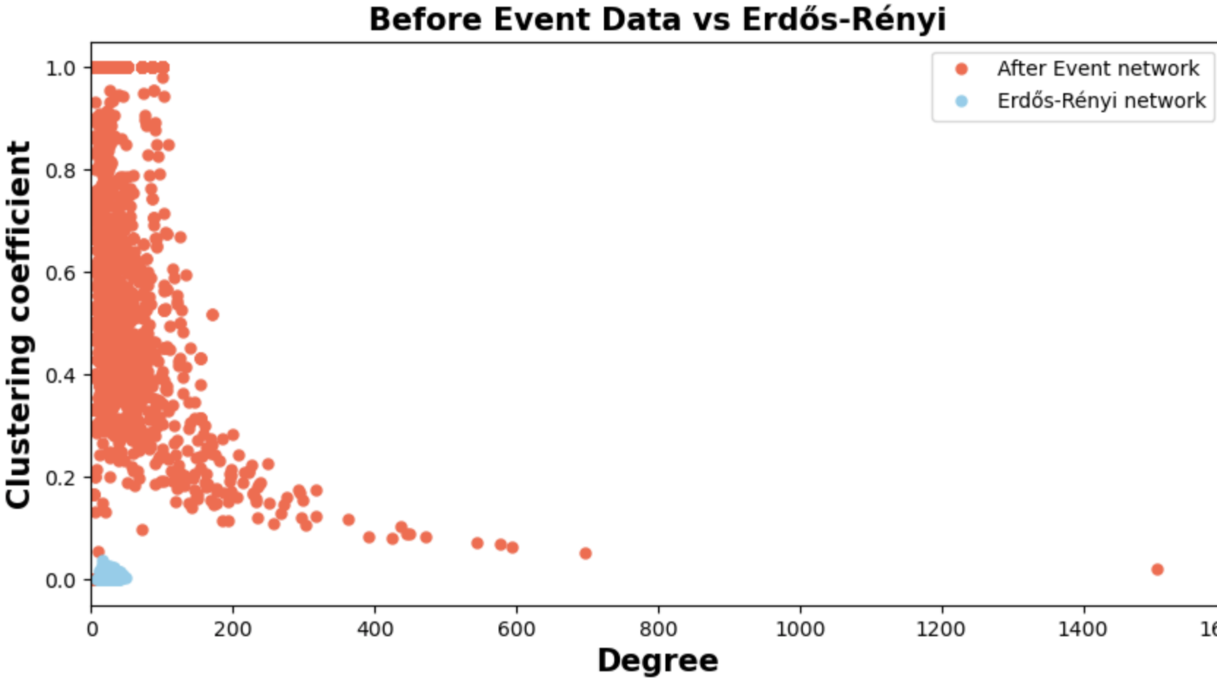
RESEARCH QUESTION: Are the nodes central because they produce spam or because of engagement?

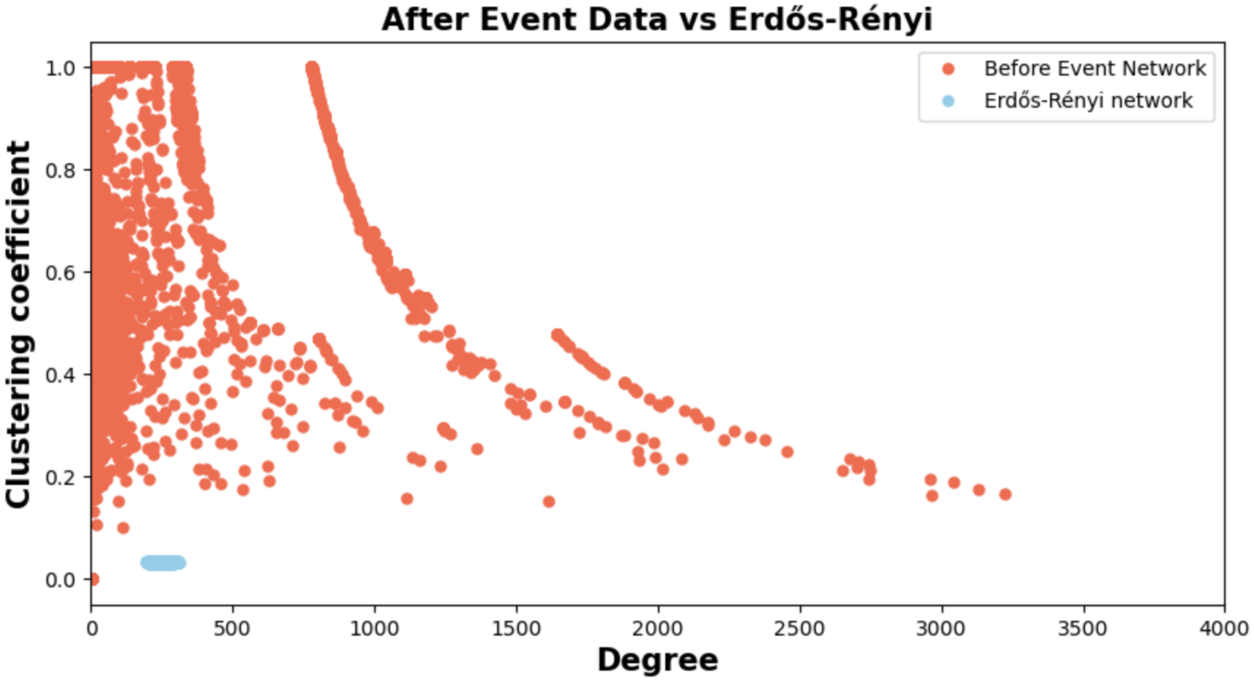
```
# Chosen centrality measure: Eigenvector
top_central_nodes_before = node_e_centr_before[:100]
top_central_nodes_after = node_e_centr_after[:100]

# Extract node IDs
top_central_node_ids_before = [node[0] for node in top_central_nodes_before]
top_central_node_ids_after = [node[0] for node in top_central_nodes_after]

# Count how many of the top 100 nodes have spam_count > 0
spam_creators_count_before = sum(1 for node in top_central_node_ids_before if g_before.nodes[node].get('spam_count', 0) > 0)
spam_creators_count_after = sum(1 for node in top_central_node_ids_after if g_after.nodes[node].get('spam_count', 0) > 0)

print(f'Spam producers (at least 1 spam message) in the top {len(top_central_nodes_before)} central nodes: {spam_creators_count_before} ({(spam_creators_count_before/len(top
print(f'Spam producers (at least 1 spam message) in the top {len(top_central_nodes_after)} central nodes: {spam_creators_count_after} ({(spam_creators_count_after/len(top_ce
```

```
Spam producers (at least 1 spam message) in the top 100 central nodes: 28 (28.00%)
Spam producers (at least 1 spam message) in the top 100 central nodes: 21 (21.00%)
```

Network Science – Transitivity

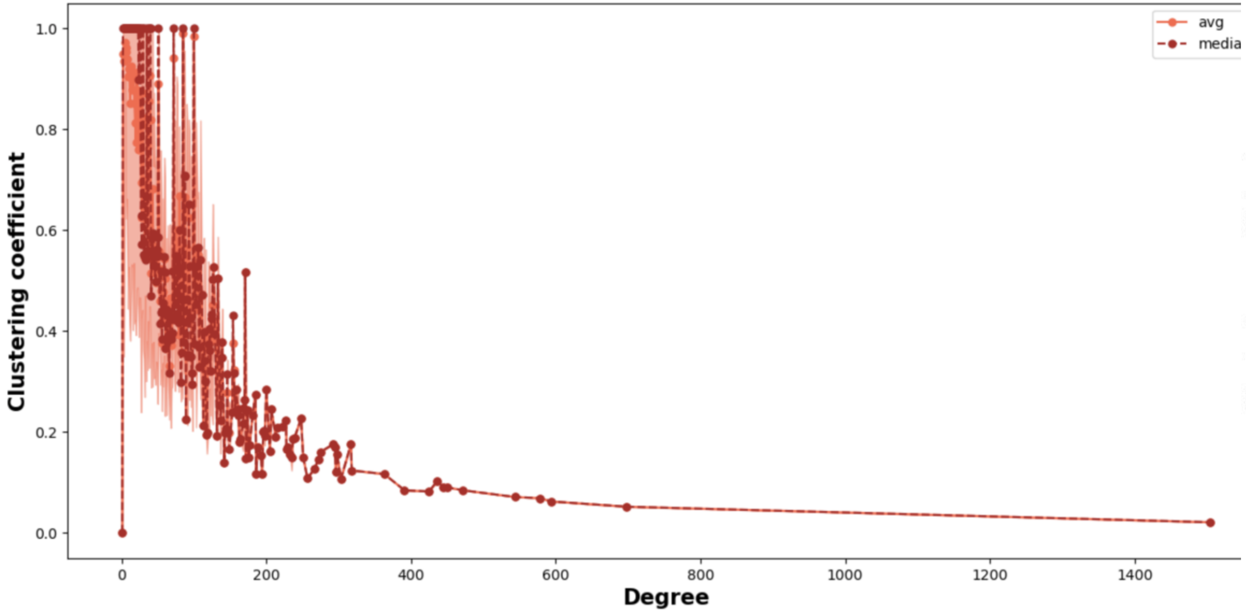
```
Transitivity before the event: 0.324423206175201
Transitivity after the event: 0.6902537241087038
```

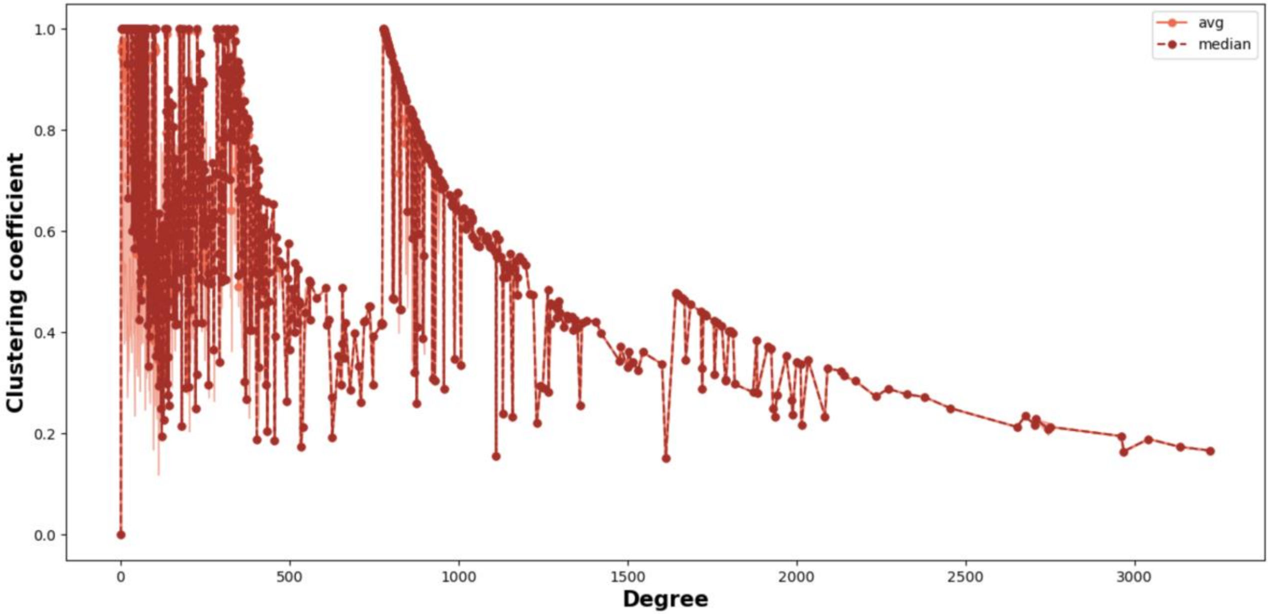




Network Science – Transitivity

```
Transitivity before the event: 0.324423206175201
Transitivity after the event: 0.6902537241087038
```





Network Science – Attribute Assortativity

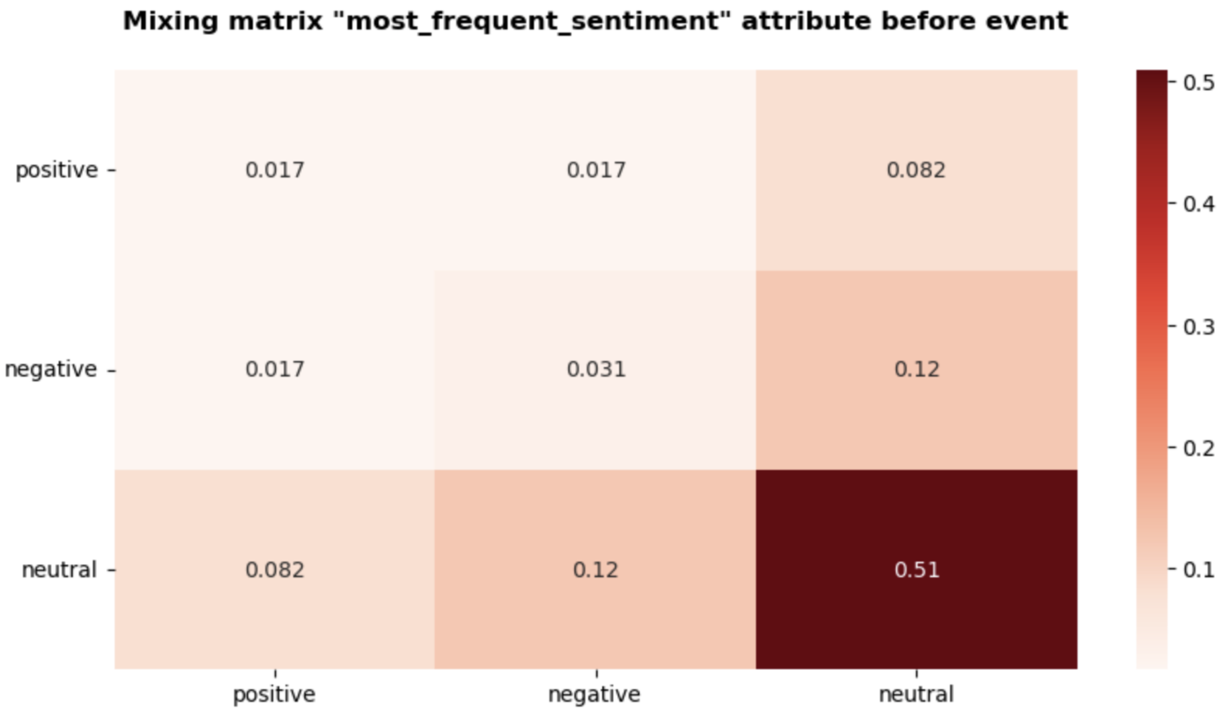
RESEARCH QUESTION: Are spam generator accounts likely to comment on the same videos (target similar videos)? What about anomalous accounts? what about accounts with similar sentiment behaviour?

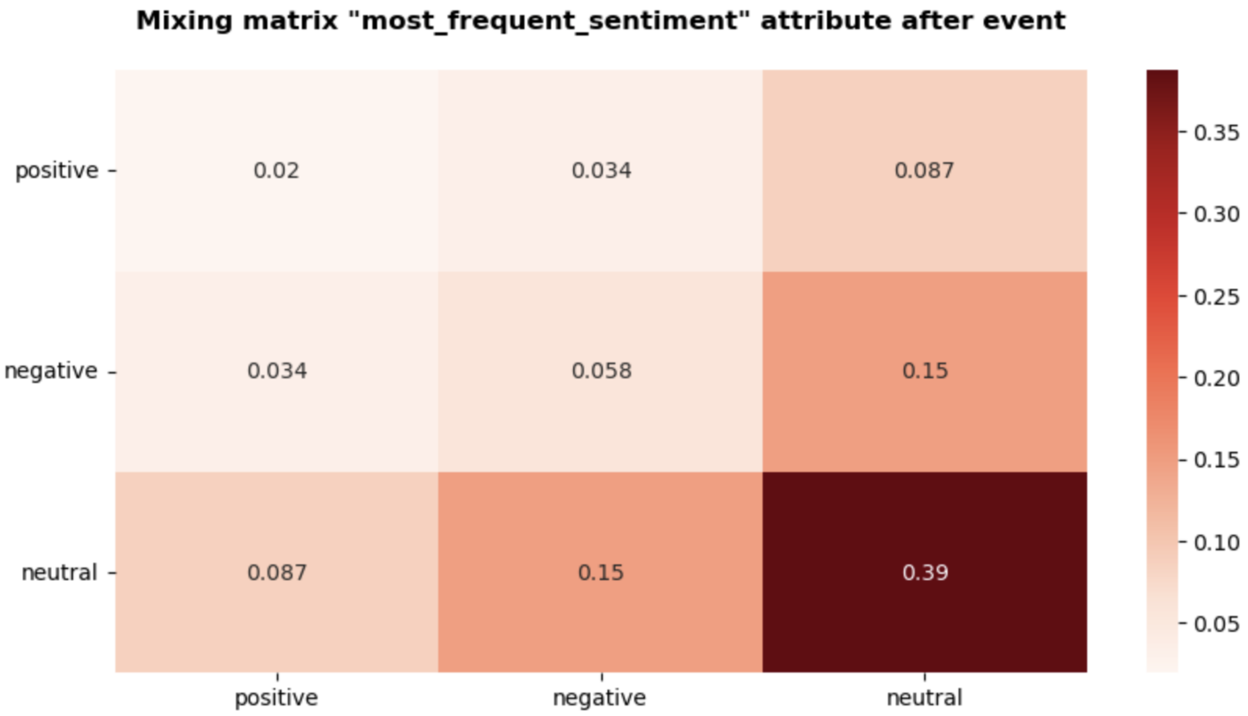
```
Before event -> Attribute assortativity for "most_frequent_sentiment": 0.012042218057111665
After event -> Attribute assortativity for "most_frequent_sentiment": 0.0026089758633647702
```

```
Before event -> Attribute assortativity for "is_spam_creator": 0.032345245503425143
After event -> Attribute assortativity for "is_spam_creator": 0.0021472813611196203
```

```
Before event -> Attribute assortativity for "is_anomalous": -0.016925227198739467
After event -> Attribute assortativity for "is_anomalous": -0.0022852026211761327
```

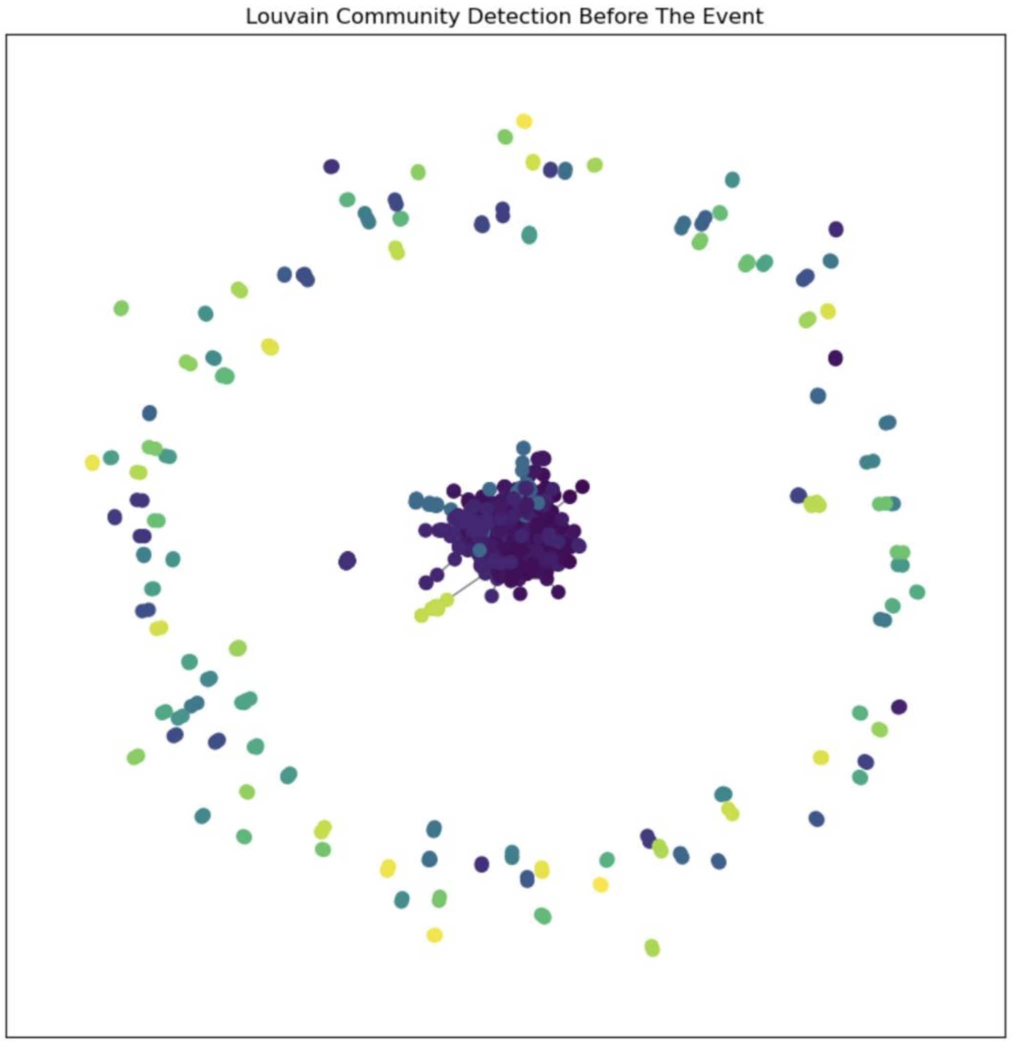
Network Science – Assortativity Mixing Matrix



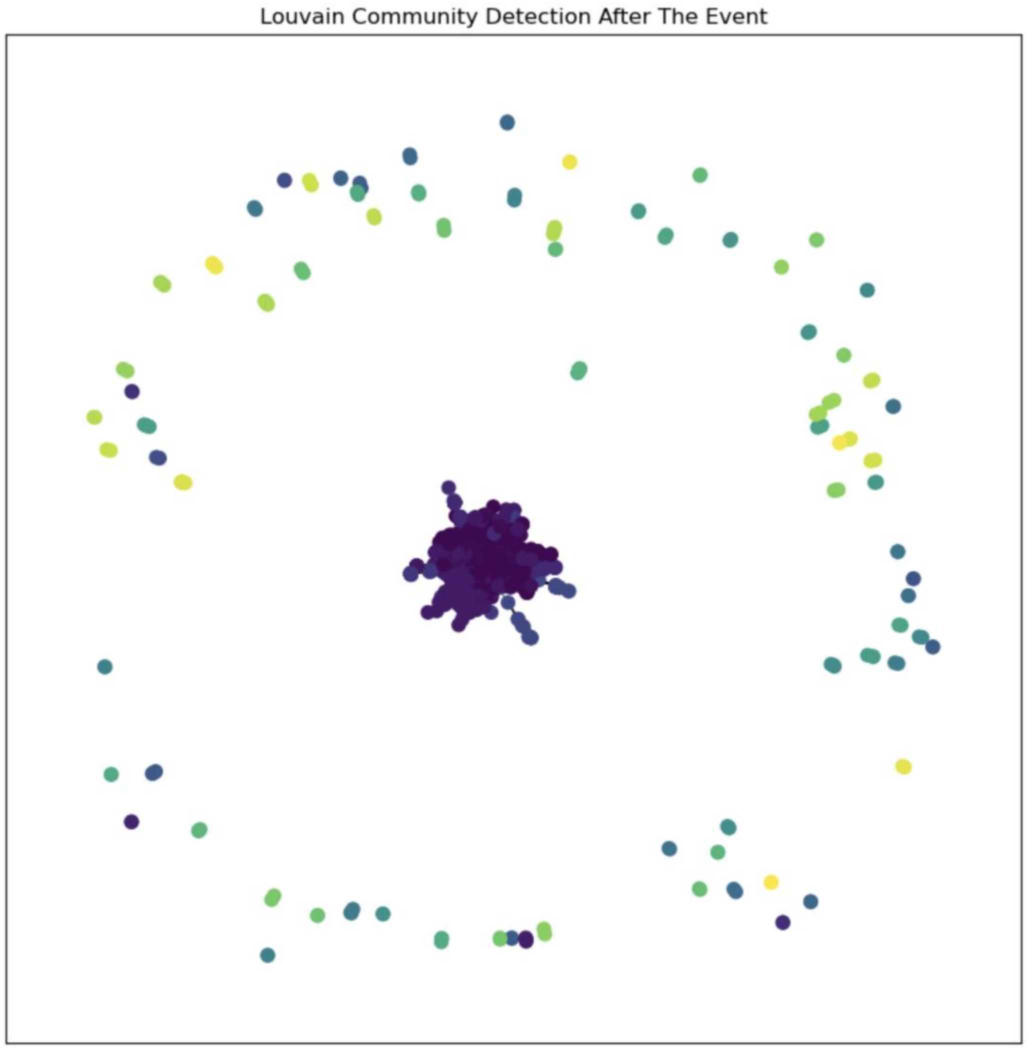


Network Science – Community Detection

Modularity: 0.6004971897940627
Number of detected communities before the event: 128



Modularity: 0.4915642815387966
Number of detected communities after the event: 99



Data Analysis – Communities spam/sentiment analysis

RESEARCH QUESTION: Is the spam percentage in the biggest communities higher than the general spam percentage?

RESEARCH QUESTION: How is the sentiment distributed across the biggest communities?

Network Science – Communities spam/sentiment analysis

```
Spam and Sentiment percentages in the top 10 communities before the event:

Community 4 (Size: 1010):
  Spam Percentage: 5.37%
  Positive: 11.58%
  Negative: 20.92%
  Neutral: 67.50%
Community 2 (Size: 714):
  Spam Percentage: 5.41%
  Positive: 12.05%
  Negative: 21.48%
  Neutral: 66.47%
Community 6 (Size: 519):
  Spam Percentage: 2.07%
  Positive: 8.03%
  Negative: 20.71%
  Neutral: 71.26%
Community 1 (Size: 366):
  Spam Percentage: 3.32%
  Positive: 9.10%
  Negative: 20.75%
  Neutral: 70.15%
Community 8 (Size: 310):
  Spam Percentage: 6.16%
  Positive: 10.27%
  Negative: 23.01%
  Neutral: 66.72%
Community 3 (Size: 223):
  Spam Percentage: 4.75%
  Positive: 8.46%
  Negative: 21.51%
  Neutral: 70.03%
Community 9 (Size: 195):
  Spam Percentage: 4.08%
  Positive: 18.42%
  Negative: 11.71%
  Neutral: 69.87%
Community 12 (Size: 194):
  Spam Percentage: 5.89%
  Positive: 15.06%
  Negative: 13.58%
  Neutral: 71.36%
Community 0 (Size: 103):
  Spam Percentage: 3.65%
  Positive: 6.85%
  Negative: 20.55%
  Neutral: 72.60%
Community 10 (Size: 80):
  Spam Percentage: 5.43%
  Positive: 4.07%
  Negative: 26.70%
  Neutral: 69.23%

Overall spam percentage in the graph: 4.41%
```

```
Spam and Sentiment percentages in the top 10 communities after the event:

Community 0 (Size: 2423):
  Spam Percentage: 6.22%
  Positive: 12.12%
  Negative: 22.16%
  Neutral: 65.72%
Community 1 (Size: 1747):
  Spam Percentage: 4.78%
  Positive: 10.57%
  Negative: 20.48%
  Neutral: 68.96%
Community 2 (Size: 820):
  Spam Percentage: 6.18%
  Positive: 10.89%
  Negative: 23.24%
  Neutral: 65.87%
Community 4 (Size: 756):
  Spam Percentage: 6.21%
  Positive: 11.28%
  Negative: 21.92%
  Neutral: 66.80%
Community 3 (Size: 595):
  Spam Percentage: 3.49%
  Positive: 10.42%
  Negative: 22.80%
  Neutral: 66.78%
Community 5 (Size: 519):
  Spam Percentage: 6.51%
  Positive: 10.13%
  Negative: 24.86%
  Neutral: 65.01%
Community 6 (Size: 514):
  Spam Percentage: 6.92%
  Positive: 15.35%
  Negative: 14.44%
  Neutral: 70.21%
Community 11 (Size: 66):
  Spam Percentage: 4.56%
  Positive: 12.86%
  Negative: 21.16%
  Neutral: 65.98%
Community 10 (Size: 49):
  Spam Percentage: 5.85%
  Positive: 15.20%
  Negative: 20.47%
  Neutral: 64.33%
Community 15 (Size: 23):
  Spam Percentage: 1.85%
  Positive: 5.56%
  Negative: 12.96%
  Neutral: 81.48%

Overall spam percentage in the graph: 4.70%
```