Distributed Sentiment Analysis of Playlists

Delaney Oliver
Zhongyuan Zheng
Gabe Roeloffs
Sean Nesbit

03/11/2021

Goals

- Understand and practice fundamental distributed computing practices
- Work collaboratively on a large dataset in databricks
- Analyze public Spotify playlist datasets to determine sentiment
- Compare Apache Spark’s approach to a Naive, non-distributed approach

The Million Playlist Dataset

- One million user generated playlists from 2010 - 2017
- Massive dataset of Playlist Names, Track Names, and Album Names
- Used in a 2018 Spotify competition where 400+ teams competed to create a model to predict the next song in a playlist
- Modified for our use in playlist sentiment analysis

Initial Testing - Looking at the Data

- Divided One Million Playlist dataset into 1000-playlist slices
- Utilized Google Colab to easily share and process small amounts of data


```
"playlists": [
    {
        "name": "Throwbacks",
        "collaborative": "false",
        "pid": 0,
        "modified_at": 1493424000,
        "num_tracks": 52,
        "num_albums": 47,
        "num_followers": 1,
        "tracks": [
            {
                "pos": 0,
                "artist_name": "Missy Elliott",
                "track_uri": "spotify:track:0UaMYEvWZi0ZqiDOoHU3YI",
                "artist_uri": "spotify:artist:2wIVse2owClT7go1WT98tk",
                "track_name": "Lose Control (feat. Ciara & Fat Man Scoop)",
                "album_uri": "spotify:album:6vV5UrXcfyQD1wu4Qo2I9K",
                "duration_ms": 226863,
                "album_name": "The Cookbook"
```

Initial Testing - Cleaning the data

- Cleaned the initial data
- Formatted 3 columns: Track Name, Column Name, Playlist Name

```
[ ]  def remove_nonascii(df, column, parentheses):
       cleaned = []
       for tweet in df[column]:
           encoded_string = tweet.encode("ascii", "ignore")
           decode_string = encoded_string.decode()
           decode_string = re.sub(r'^https?:\/\/.*[\r\n]*', '', decode_string, flags=re.MULTILINE)
           decode_string = re.sub(r'http\S+', '', decode_string)
           if parentheses:
             decode_string = re.sub(r' ?\([^)]+\)', '', decode_string)

           cleaned.append(decode_string)
       return cleaned
```

```
[ ]  all_data['cleaned_track'] = remove_nonascii(all_data, 'track_name',  1)
     all_data['cleaned_album'] = remove_nonascii(all_data, 'album_name', 1)
     all_data['cleaned_playlist'] = remove_nonascii(all_data, 'Playlist', 0)
```

Initial Testing - First Sentiment Analysis

- Associated a sentiment analysis using the open-source TextBlob library.
- TextBlob is founded on NLTK and Pattern, a NLP library.

```
def sentiment_avg(row, col):
  blob = TextBlob((row[col]))
  num_sentences = 0
  sentiment = 0
  for sentence in blob.sentences:
    num_sentences += 1
    sentiment += sentence.sentiment.polarity
  if num_sentences == 0:
    out = 0
  else:
    out = sentiment/num_sentences
  return out
```

```
[ ]  start = time.time()

     all_data["track_sentiment"] = all_data.apply(lambda x: sentiment_avg(x, 'cleaned_track'), axis=1)
     all_data["album_sentiment"] = all_data.apply(lambda x: sentiment_avg(x, 'cleaned_album'), axis=1)
     all_data["playlist_sentiment"] = all_data.apply(lambda x: sentiment_avg(x, 'cleaned_playlist'), axis=1)
```

Initial Testing - Results

- Initial Data Before Aggregating by Playlist

	pos	artist_name	track_name	duration_ms	album_name	Playlist	cleaned_track	cleaned_album	cleaned_playlist	track_sentiment	album_sentiment	playlist_sentiment
0	0	Missy Elliott	Lose Control (feat. Ciara & Fat Man Scoop)	3.781050	The Cookbook	Throwbacks	Lose Control	The Cookbook	Throwbacks	0.00	0.00	0.0
1	1	Britney Spears	Toxic	3.313333	In The Zone	Throwbacks	Toxic	In The Zone	Throwbacks	0.00	0.00	0.0
2	2	Beyoncé	Crazy In Love	3.932217	Dangerously In Love (Alben für die Ewigkeit)	Throwbacks	Crazy In Love	Dangerously In Love	Throwbacks	-0.05	0.50	0.0
3	3	Justin Timberlake	Rock Your Body	4.454433	Justified	Throwbacks	Rock Your Body	Justified	Throwbacks	0.00	0.40	0.0
4	4	Shaggy	It Wasn't Me	3.793333	Hot Shot	Throwbacks	It Wasn't Me	Hot Shot	Throwbacks	0.00	0.25	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...
39	39	James Arthur	Say You Won't Let Go	3.524433	Back from the Edge	thinking of you	Say You Won't Let Go	Back from the Edge	thinking of you	0.00	0.00	0.0
40	40	Big Words	The Answer	4.394650	Hollywood, a Beautiful Coincidence	thinking of you	The Answer	Hollywood, a Beautiful Coincidence	thinking of you	0.00	0.85	0.0
41	41	Allan Rayman	25.22	3.153550	Roadhouse 01	thinking of you	25.22	Roadhouse 01	thinking of you	0.00	0.00	0.0
42	42	Jon Jason	Good Feeling	3.245333	Good Feeling	thinking of you	Good Feeling	Good Feeling	thinking of you	0.70	0.70	0.0
43	43	Grizfolk	Cosmic Angel - Acoustic From Capitol Studios	4.286567	Cosmic Angel	thinking of you	Cosmic Angel - Acoustic From Capitol Studios	Cosmic Angel	thinking of you	0.00	0.00	0.0

67503 rows × 12 columns

Parallelized Approach - Loading Data

- After initial testing we moved the Million Playlist Dataset onto Databricks to begin proper development
- The JSON first had to be converted into a spark table

```
# read playlist json files to dataframe and save dataframe to spark table name "playlists"
def transform_json_to_dataframe():
  playlist_df = spark.read.option("multiline","true").json(DATASET_PATH)
  playlist_df.show(1)
  playlist_df = playlist_df.select("info.version", explode("playlists"))
  playlist_df = playlist_df.select("col.name", "col.tracks").withColumnRenamed("name", "playlist_name")
  playlist_df.show(1)

  concat_track_names = concat(playlist_df.tracks.track_name).alias("track_names")
  concat_album_names = concat(playlist_df.tracks.album_name).alias("album_names")
  playlist_df = playlist_df.select("playlist_name", concat_track_names, concat_album_names)
  playlist_df.write.mode("overwrite").saveAsTable("playlists");

# execute the function
transform_json_to_dataframe()
```

Parallelized Approach - Analysis Functions

- We then utilized a simple map() function for sentiment analysis on a sentence
- These functions will then be parallelized by the SparkContext

```
# analyze single sentence
def analyze_single(text):
    blob = TextBlob(text)
    polarity, subjectivity = blob.sentiment
    return polarity

# analyze sentence list
def analyze_multiple(texts):
    if texts is None or len(texts) == 0:
        return 0.0

    scores = map(analyze_single, texts)
    return list(scores)

# Analyzes a list of album or track names.
def analyze_aggregate(texts):
  scores = analyze_multiple(texts)

  if len(scores) == 0:
    return 0.0
```

Parallelized Approach - Execution

- To generate the scored playlists, we operate on the table with our sentiment functions
- This generates new columns with valuable data for analysis

```
def generate_scored_playlists():
    analyze_single_udf = udf(analyze_single, FloatType())
    analyze_multiple_udf = udf(analyze_multiple, ArrayType(FloatType()))
    analyze_aggregate_udf = udf(analyze_aggregate, FloatType())

    scored_df = plain_df.withColumn("p_score", analyze_single_udf("playlist_name"))
    scored_df = scored_df.withColumn("t_scores", analyze_multiple_udf("track_names"))
    scored_df = scored_df.withColumn("a_scores", analyze_multiple_udf("album_names"))
    scored_df = scored_df.withColumn("agg_score", analyze_aggregate_udf("track_names"))
```

So what does our data look like now?

Table before scoring sentiments:

playlist_name	track_names	album_names
Throwbacks	[Lose Control (fe...	[The Cookbook, In...
Awesome Playlist	[Eye of the Tiger...	[Eye Of The Tiger...
korean	[Like You, GOOD (...	[On And On, GOOD ...
mat	[Danse macabre, P...	[French Festival,...
90s	[Tonight, Tonight...	[Mellon Collie an...

only showing top 5 rows

Table after scoring sentiments:

playlist_name	track_names	album_names	p_score	t_scores	a_scores	agg_score
Throwbacks	[Lose Control (fe...	[The Cookbook, In...	0.0	[0.0, 0.0, -0.05,...	[0.0, 0.0, 0.5, 0...	0.079933606
Awesome Playlist	[Eye of the Tiger...	[Eye Of The Tiger...	1.0	[0.0, 0.0, 0.0, 0...	[0.0, 0.0, 0.0, 0...	0.031684984
korean	[Like You, GOOD (...	[On And On, GOOD ...	0.0	[0.0, 0.7, -0.3, ...	[0.0, 0.7, -0.3, ...	0.03796503
mat	[Danse macabre, P...	[French Festival,...	0.0	[0.0, -0.05, 0.0,...	[0.0, 0.0, 0.0, 0...	0.024690885
90s	[Tonight, Tonight...	[Mellon Collie an...	0.0	[0.0, 0.0, 0.0, -...	[0.0, 0.0, 0.0, 0...	0.012815126

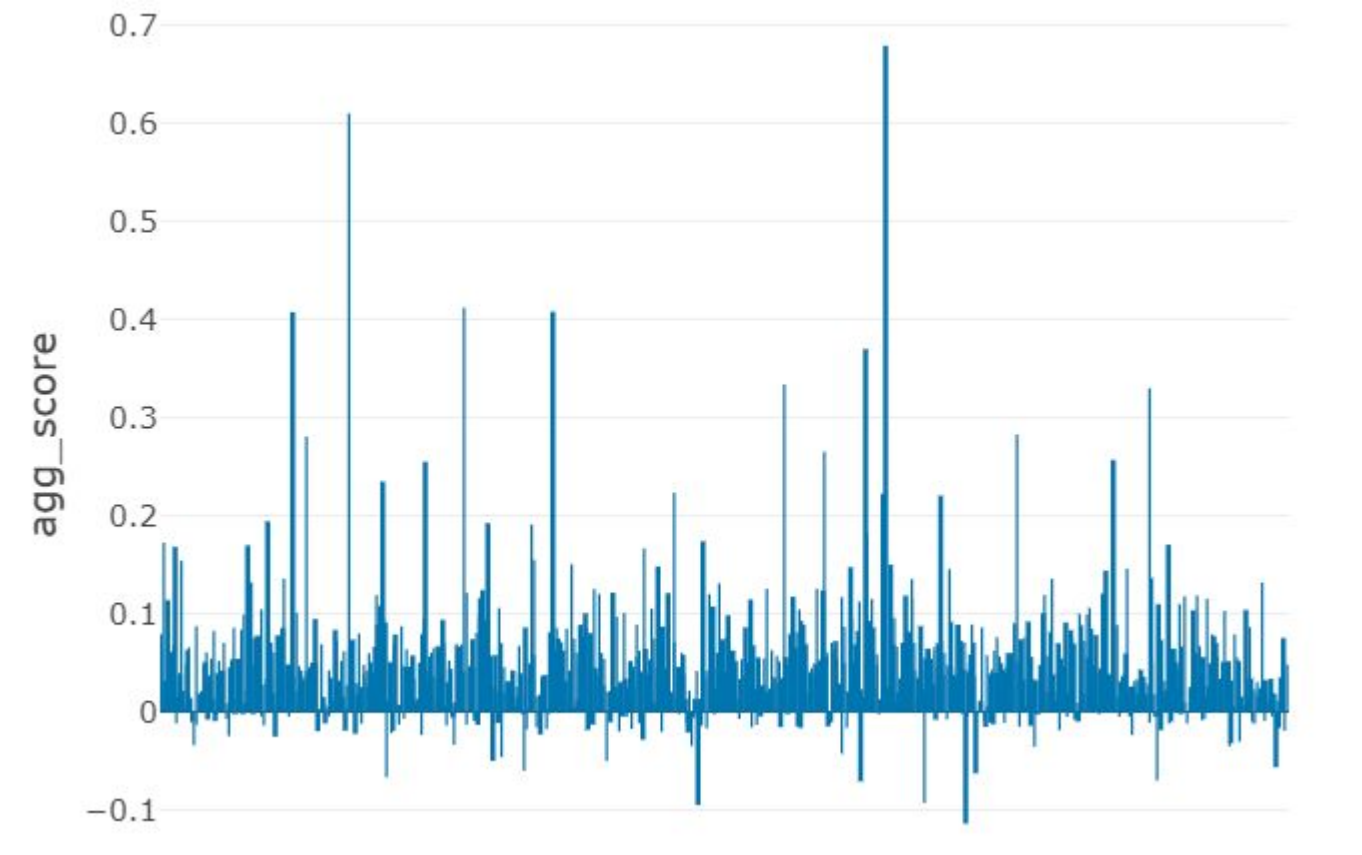
Parallelized Approach Results

- The cluster already contains a spark table of the data named “playlists”
- This allows us to bypass timing the read functions and focus on the transformations
- 2886.55s / 60 = **48.11 minutes** for **100 files (100k playlists)**

```
1  dbutils.fs.rm("dbfs:/user/hive/warehouse/scored_playlists", recurse=True)
2  start = time.time()
3  generate_scored_playlists()
4  end = time.time()
5  print(f"The runtime of the Spark approach is: {end-start} seconds.")
```

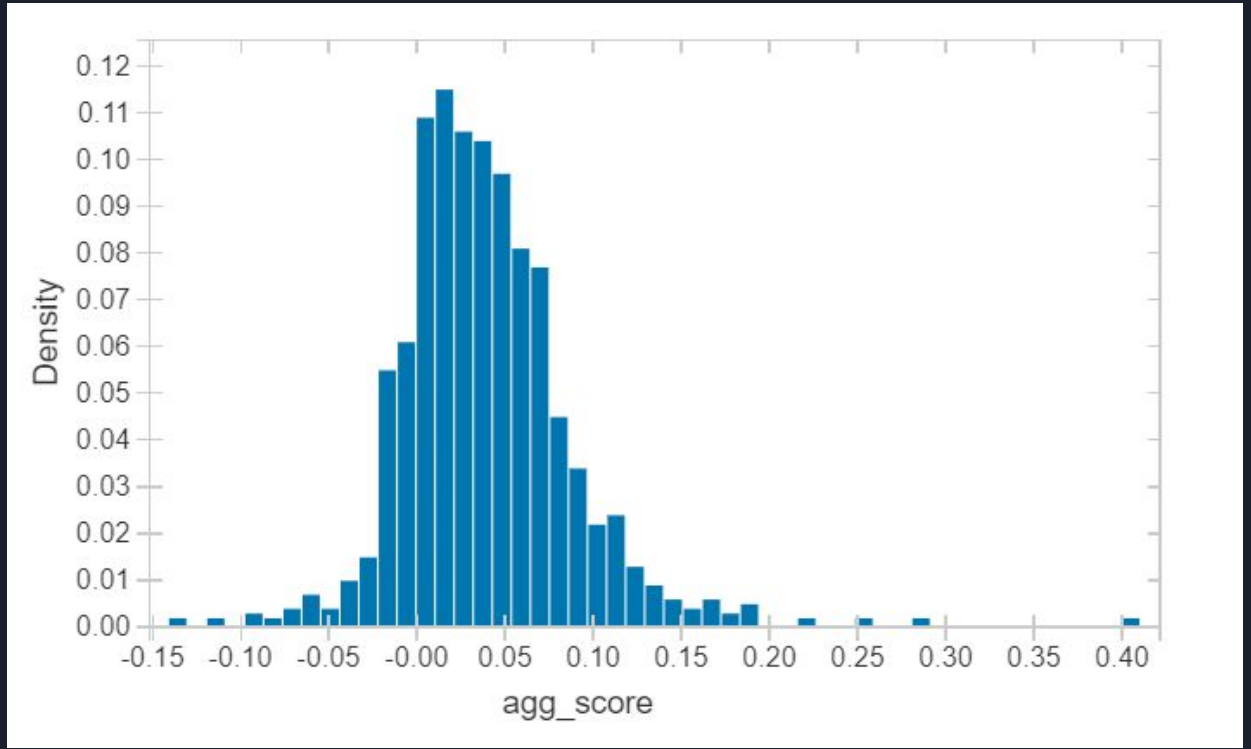
```
The runtime of the Spark approach is: 2886.5508971214294 seconds.
```

Aggregate Score Distributions



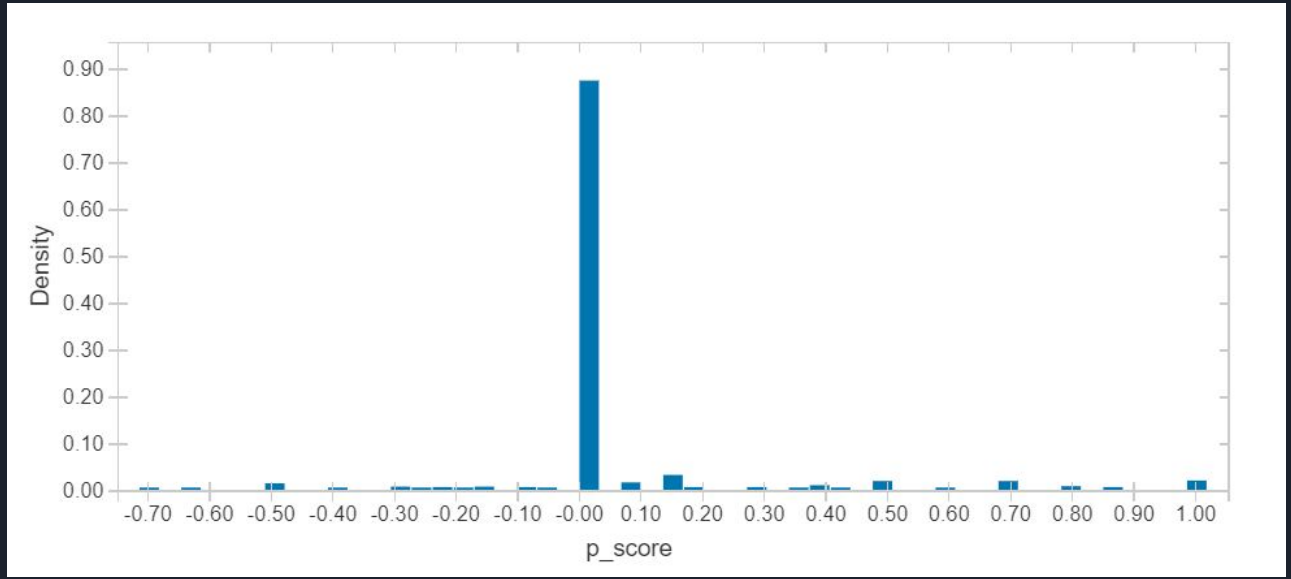
Histogram of Aggregate Scores

- Most aggregate sentiment scores fell between 0.0 and 0.10
- Few aggregate scores fell below 0.0
- Most playlists have a slight positive sentiment



Histogram of Playlist Scores

- Playlist Score analyzes the sentiment of a playlist by the playlist name alone
- We wanted to attempt to see if there was a correlation between a short playlist title and the aggregate score
- These playlists titles were often too short to have any tangible sentiment associated



Summary of Distributed Results

- Our analysis found most playlists have a slight positive Aggregate Score
- Playlist Score was not a good predictor of Aggregate Score
- Very few playlists have negative scores (Playlist or Aggregate)
- With more attributes we could make a finer-grain analysis of each track and approach the true sentiment

Naive Approach

What does the Naive data look like?

cleaned_playlist	cleaned_track	cleaned_album	album_sentiment	agg_score
	[Natural Born Killer, Diamond Eyes, Rose Of Sh...	[Nightmare, Diamond Eyes, The End Of Heartache...	[0.0, 0.0, 0.35714285714285715, 0.0, 0.1785714...	0.012488
CHiLl	[Make Me, Party Monster, Don't Wanna Know, Let...	[Make Me, Starboy, Red Pill Blues, Encore, Chi...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.6, 0.0, 0.0,...	0.066667
Frozen	[Frozen Heart, Do You Want to Build a Snowman?...	[Frozen, Frozen, Frozen, Frozen, Froze...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	0.079015
indie rock	[Be Good, Bambi, Your English Is Good, Nature ...	[Smith, Champ, Elephant Shell, A Lesson In Cri...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	0.172222
#Relaxed	[All That I Can Say, Reminisce, Butterfly, Cha...	[Mary, What's The 411?, Butterfly, Comin' From...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.5, 0.0, 0.0, 0.0, ...	0.031220
...	...	...	...	...
xmas party	[I Don't Fuck With You, Don't Panic, Tuesday, ...	[Dark Sky Paradise, Don't Panic, I LOVE MAKONN...	[-0.15, 0.0, 0.5, 0.0, -0.25, -0.3, 0.0, 0.25,...	0.015231
xx	[Win Some, Lose Some, I Know, Deserve It, Jump...	[Dark Sky Paradise, Dark Sky Paradise, Dark Sk...	[-0.15, -0.15, -0.15, 0.0, 0.0, 0.0, 0.0, 0.0,...	0.022220
yo	[Dirty Little Secret, Since U Been Gone, I Wri...	[Move Along, Breakaway, A Fever You Can't Swea...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	0.019033
yoga	[Faith, Hanuman Baba, Easy - Music From The Mo...	[Covers, Vol. 2, Greatest Hits of the Kali Yug...	[0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	0.131481

How long does it take to score **1000** playlists?

```
start = time.time()
# creating a dataset of all the tracks in each playlist
def generate_scores_naive(playlists):
  list_data =[]
  for i in range(0,len(playlists)):
      data = pd.DataFrame(playlists['tracks'][i])
      data = data[['pos', 'artist_name', 'track_name', 'duration_ms', 'album_name']]
      data["Playlist"] = playlists['name'][i]

      list_data.append(data)

  all_data = pd.concat(list_data)
  all_data
  all_data['duration_ms'] = all_data['duration_ms'] / 60000
  all_data.rename(columns = {'duration_ms':"duration"}

  #assign sentiment scores to track, album and playlist name columns
  all_data["track_sentiment"] = all_data.apply(lambda x: sentiment_avg(x, 'track_name'), axis=1)
  all_data["album_sentiment"] = all_data.apply(lambda x: sentiment_avg(x, 'album_name'), axis=1)
  all_data["playlist_sentiment"] = all_data.apply(lambda x: sentiment_avg(x, 'Playlist'), axis=1)

  #createing aggregate score of all tracks in a playlist for overall "mood" of each playlist
  scored_data = all_data.groupby("Playlist").agg(
    {"track_name": lambda x: x.tolist(),
     "album_name":  lambda x: x.tolist(),
     "album_sentiment":  lambda x: x.tolist(),
     "track_sentiment": "mean"
    })
  scored_data = scored_data.rename(columns = {'track_sentiment': "agg_score"})
  return scored_data

playlists1_scored = generate_scores_naive(playlists1)
end = time.time()
print(f"Runtime of the program is {end - start}")
```

```
Runtime of the program is 79.09670662879944
Command took 1.32 minutes -- by dcoliver@calpoly.edu at 3/11/2021, 1:01:51 PM on My Cluster
```

Command took 1.32 minutes

Time Comparisons between Naive vs. Distributed

Running the **Naive Approach** for each of the 100 files :

- 1.32min *100 files = 132 minutes or **2.2 hours**

Running the distributed approach on 100 files:

- **48 .11 minutes**

If we considered the entire dataset, which contains 1 million playlists, or 1000 files:

- **Naive (assuming linear time growth):** 2.2 hours * 10 = 22 hours
- **Distributed Approach :** 48.11 minutes * 10 = 8.018 hours

This is a **63.55%** decrease in time to completely analyze all 1 million playlists in the original dataset.

Proposed Future Goals

- Implement K-Nearest-Neighbors to classify the playlists into positive and negative sentiments
  - First using SciKit-Learn for the ease of implementation

- Implement other distributed approaches such as Ray to determine relative speeds on this dataset

- Flatmap the data, remove duplicates, and analyze Track Score sentiments against other sentiment variables

Thanks!