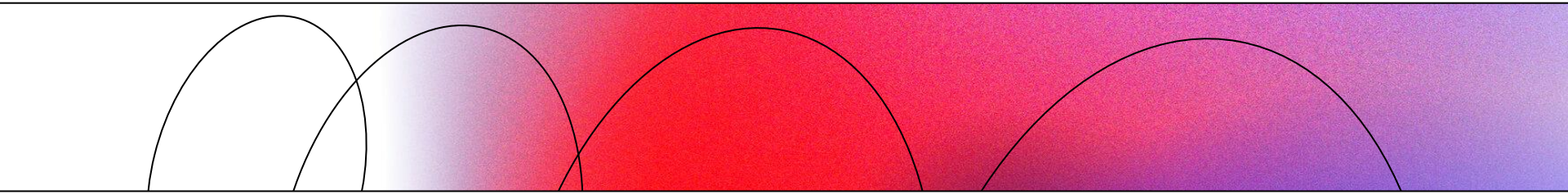
Is “Good” Music Biased?



Predicting Grammy Winning Songs

Robbery


Back (1133) Kendrick Real Contact
Today 8:53 PM
You got robbed. I wanted you to win. You should have. It's weird and sucks that I robbed you. I was gonna say that during the speech. Then the music started playing during my speech and I froze. Anyway, you know what it is. Congrats on this year and your music. Appreciate you as an artist and as a friend. Much love


GETTY IMAGES

Aim & Motivation

Why?

- We both love music!
- We both wondered why some of our favorite artists don't take home the grand old prize (Kendrick Lamar's "To Pimp a Butterfly")
- Controversy!

Goals:

- Investigate possible biases in the music industry
- See what goes into the selection process for Grammy-winning albums.
- Predict whether an album will receive a Grammy.
    - No Genres, solely pure statistics (more later)!
- Contribute to a more transparent and fair music industry.

Data collection & cleaning

Web-scraping

- Wikipedia - Grammy info from 3 most recent decades
- MusicBrainz - specific information about artist(s) (gender, type of band, etc.)
- Spotify API - musical data

```
[ ] # Inputting input credentials
    cid = 'b769747d16a348039e288a81d023e6be'
    secret = '736c6a4a0e2b4345a4344ae3a07fbf25'
    client_credentials_manager = SpotifyClientCredentials(client_id=cid, client_secret=secret)
    sp = spotipy.Spotify(client_credentials_manager=client_credentials_manager)

[ ] df_spotify = pd.read_csv("/content/df_wiki.csv")

    # Find queries for easy to search albums on Spotify
    artists, album_ids, album_uris, list_no_tracks, list_release_dates = [], [], [], [], []

    for i in range(len(df_spotify['album_name'])):

      query = "album: " + df_spotify["album_name"][i] + " artist: " + df_spotify["artists"][i]

      result_list = sp.search(query, limit = 1, type='album', market='US')['albums']['items']
      if result_list:
        result = result_list[0]
        album_ids.append(result['id'])
        album_uris.append(result['uri'])
        list_no_tracks.append(result['total_tracks'])
        list_release_dates.append(result['release_date'])
        print(df_spotify['album_name'][i])
      else: # For error catching
        # Find queries for the more difficult to search albums on Spotifiy by instead only looking for album name
        print("No results found for query: ", query,". Using brute force instead.")  # Used to fix mistakes
        print(sp.search(q=df_spotify["album_name"][i], type="album", limit=2))

    Supernatural
```

- Accessing Spotify's API to get grammy winning albums statistics, which we previously collected album names and artist(s) for.

- Getting specific features about each album:
  - key
  - loudness
  - speechiness
  - acousticness
  - liveness

```
list_means = []
list_vars = []

for tracklist in df_spotify["track_id_list"]:
  list_dicts = []
  for track in tracklist:
    feature_dict = sp.audio_features(track)[0]
    list_dicts.append(feature_dict)

  df_track_feats = pd.DataFrame(list_dicts, index = tracklist)
  df_track_feats = df_track_feats.drop(columns = ['type', 'id', 'uri', 'analysis_url', 'track_href'])
  means = df_track_feats.iloc[:, 0:12].mean()
  list_means.append(means)
  vars = df_track_feats.iloc[:, 0:12].var()
  list_vars.append(vars)
list_means
```

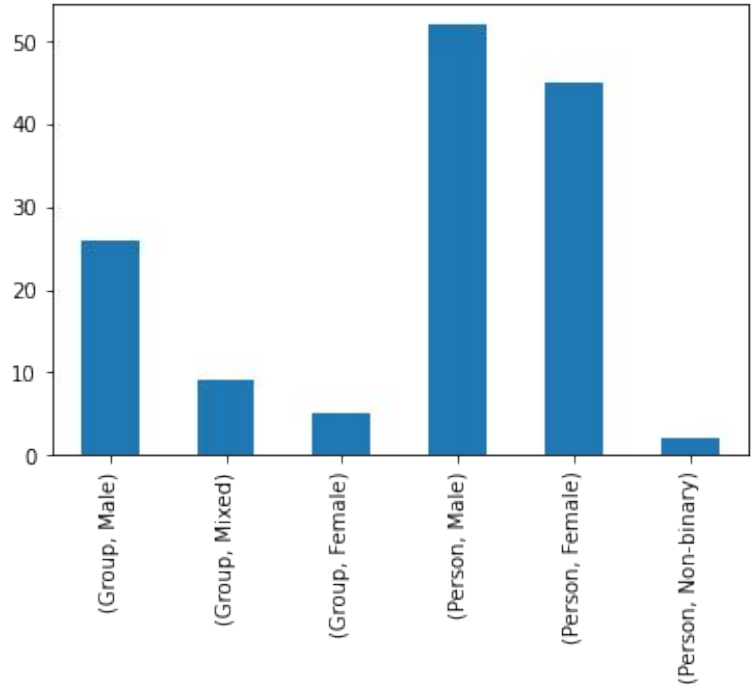
Combined DataFrame

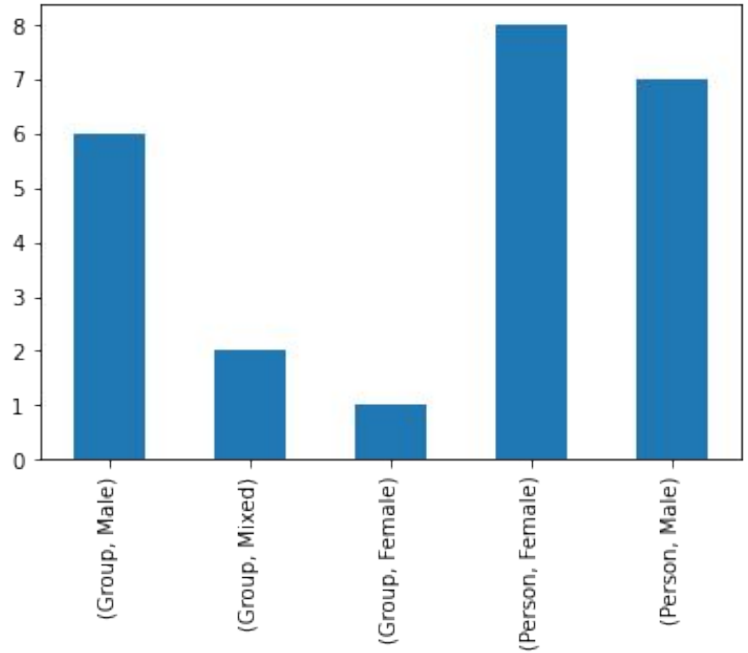
	artists	type	sex	album_name	won	num_of_tracks	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	duration_ms
0	Santana	Group	Male	Supernatural	True	14	0.582786	0.792071	7.071429	-6.360286	0.428571	0.059243	0.206586	0.191212	0.179429	0.734000	112.041000	319684.714286
1	TLC	Group	Female	FanMail	False	17	0.691765	0.620059	5.764706	-7.341235	0.529412	0.159865	0.073153	0.021650	0.213459	0.607824	114.408765	223631.294118
2	The Chicks	Group	Female	Fly	False	13	0.593846	0.591192	4.153846	-8.442769	1.000000	0.035015	0.207601	0.001895	0.154046	0.568231	129.986462	220581.461538
3	Backstreet Boys	Group	Male	Millennium	False	12	0.643000	0.618417	4.833333	-6.766333	0.500000	0.030092	0.217875	0.000001	0.184825	0.511167	120.629417	230227.750000
4	Diana Krall	Person	Female	When I Look in Your Eyes	False	13	0.532000	0.166946	5.769231	-16.167308	0.538462	0.040623	0.877846	0.017048	0.123092	0.326000	109.450923	250796.000000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
134	Coldplay	Group	Male	Music of the Spheres	False	12	0.419075	0.562008	3.583333	-11.275250	0.666667	0.053567	0.375252	0.330368	0.258567	0.216550	140.935750	209217.750000
135	Beyonce	Person	Female	Renaissance	False	16	0.729500	0.665750	4.000000	-7.248938	0.687500	0.138194	0.062996	0.003560	0.260787	0.429656	117.673000	233856.937500
136	Lizzo	Person	Female	Special	False	12	0.693000	0.672417	5.916667	-5.537500	0.416667	0.078292	0.109871	0.000001	0.217617	0.658500	116.024917	176376.916667
137	Bad Bunny	Person	Male	Un Verano Sin Ti	False	23	0.756652	0.657391	5.304348	-6.060696	0.478261	0.119965	0.258104	0.000249	0.216961	0.484826	118.279391	213529.173913
138	ABBA	Group	Mixed	Voyage	False	10	0.556200	0.473100	4.500000	-9.563000	0.900000	0.029380	0.542100	0.074080	0.178990	0.508300	108.095500	222918.500000

139 rows × 18 columns

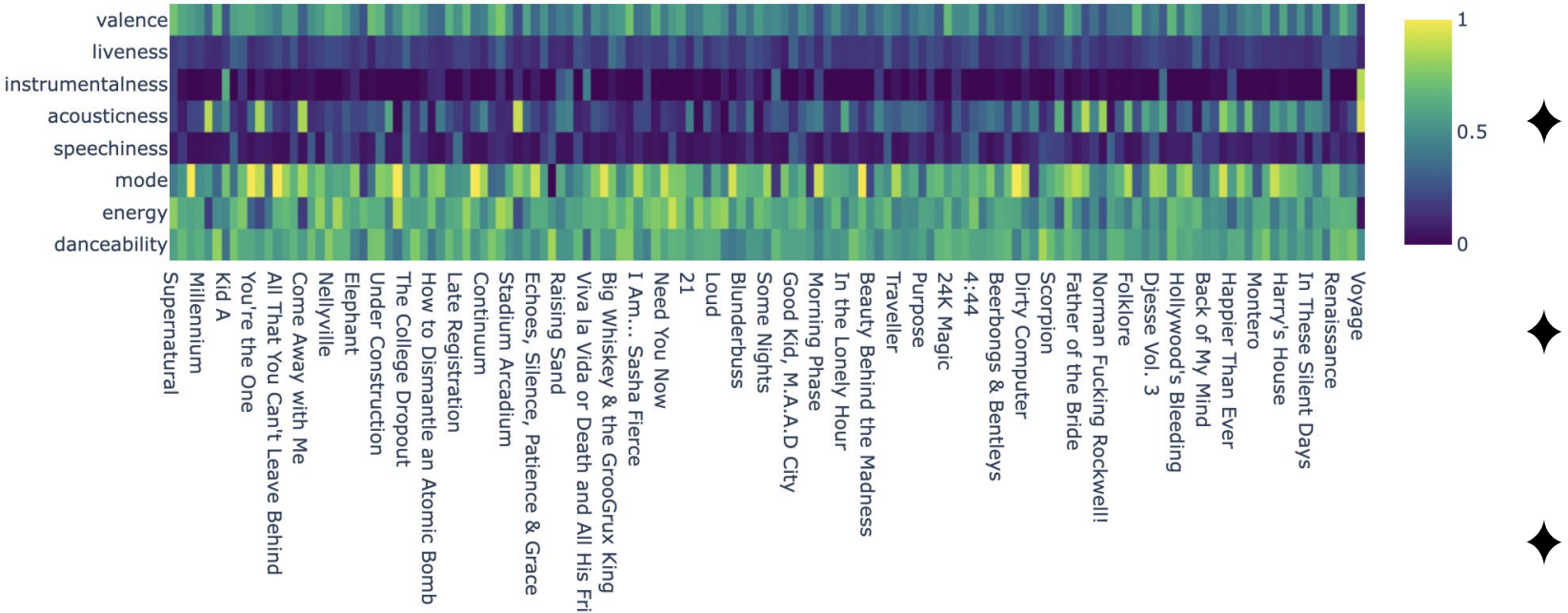
Data Exploration

Types of nominees



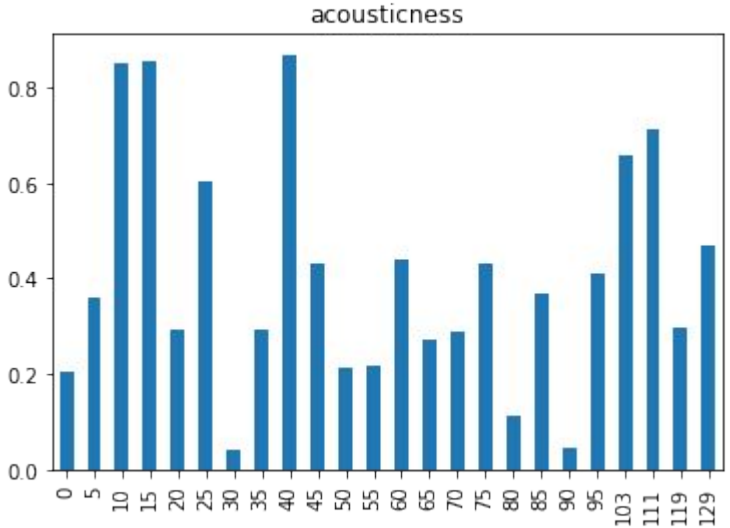


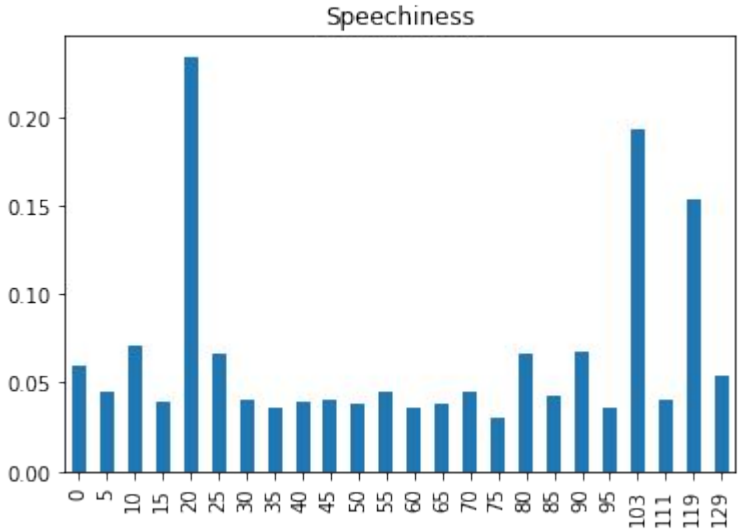
Musical features of the data



winners only

Traits of winners





Machine Learning

Creating the best model

- Grabbing best features: recursively go through every combination and grab the one with the highest f1-score

- Getting the best scaler using loop
  - for KNN: loop through the metric as well
  - `['album name', 'acousticness','sex', 'valence',`
  - `'danceability','energy', 'artists','duration ms', 'instrumentalness','key', 'liveness','loudness', 'tempo']`
- Using Grid-Search CV:
  - for KNN: getting the best K (k = 7)
  - for Decision tree parameters: criterion, splitter (“gini”, “random”)
  - `['album name', 'acousticness', 'sex', 'valence', 'danceability', 'energy', 'artists', 'instrumentalness', 'liveness', 'tempo']`

Recursive Functions

```
[ ] # Create column transformer based on desired features
    def createColTrans(features): #instead of features, call it curVars
      col_trans = []

      for feat in features:
        name = "scaler" + feat
        if (type(df_training[feat][0]) == np.float64):
          col_trans.append((name, StandardScaler(), [feat]))
        if (type(df_training[feat][0]) == str):
          col_trans.append((name, TfidfVectorizer(smooth_idf=False, norm=None), feat)) #max_features=k??

      column_transformer = ColumnTransformer(col_trans)
      return column_transformer
```

```
  # Creates modifiable column transformer based on desired scaler
    def modifyColTrans(scaler, features):
      col_trans = []

      for feat in features:
        name = "scaler" + feat
        if (type(df_training[feat][0]) == np.float64):
          col_trans.append((name, scaler, [feat]))
        if (type(df_training[feat][0]) == str):
          col_trans.append((name, TfidfVectorizer(smooth_idf=False, norm=None), feat)) #max_features=k??

      column_transformer = ColumnTransformer(col_trans)
      return column_transformer
```

```
[ ] def createPipeline(col_transformer, model):
      pipeline = make_pipeline(
          col_transformer,
          model
      )
      return pipeline
```

```
[ ] def createCrossValue(pipeline, currentVariablesModel):
      score = np.mean(cross_val_score(
        pipeline,
        X = df_training[currentVariablesModel],
        y = y_train,
        scoring="f1",
        cv=4
      ))

      return score
```

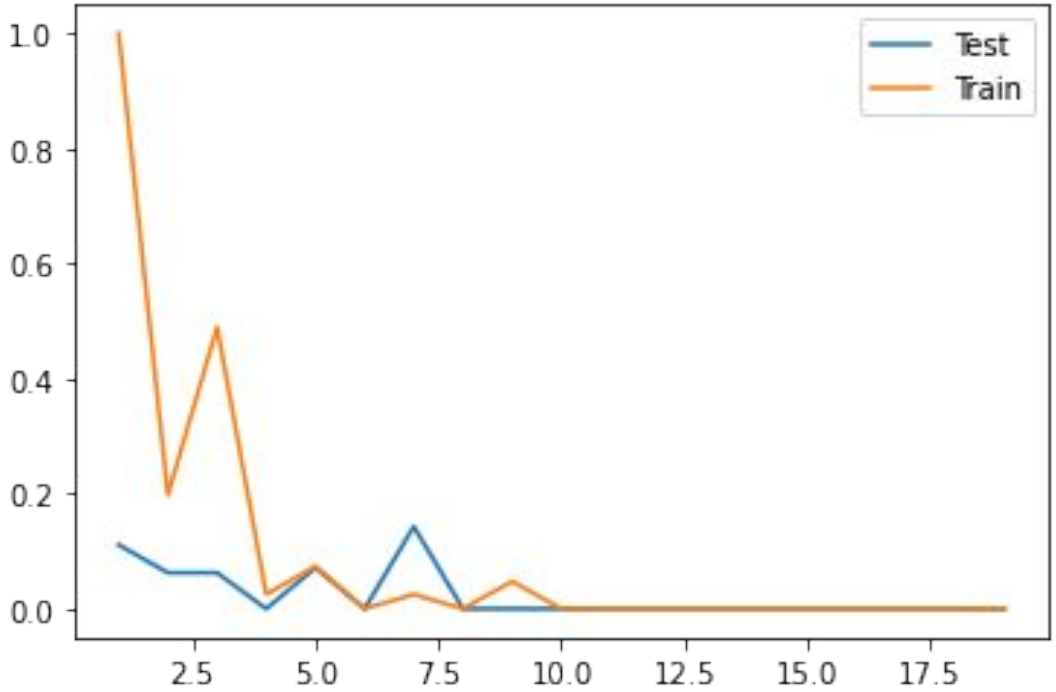
- Helper functions that we ran all our data through.

```
[ ] scalers = [StandardScaler(), Normalizer(), MinMaxScaler()]
    metrics = ["euclidean", "minkowski", "manhattan"]

    scores = []
    allCombs = []

    for scaler in scalers:
      col_trans = modifyColTrans(scaler, best_feats_knn)

      for metric in metrics:
        pipeline = make_pipeline(
          col_trans,
          KNeighborsClassifier(n_neighbors=20, metric=metric)
        )

        score = createCrossValue(pipeline, best_feats_knn)
        scores.append(score)

        allCombs.append([scaler, metric])

    # Get minimum error
    max_index = scores.index(sorted(scores)[len(scores)-1])

    # Define best pipeline
    best_scaler_knn = allCombs[max_index][0]
    best_metric_knn = allCombs[max_index][1]
    pipeline_knn = make_pipeline(
        modifyColTrans(best_scaler_knn, best_feats_knn),
        KNeighborsClassifier(n_neighbors=20, metric=best_metric_knn)
        )
```

```
[ ] allCombs[max_index]
```

- Getting best features and metrics

Graphing F1 scores for different Ks



Comparing cross-val f1-scores

0.14

KNN

K = 7
Standard Scaler, Euclidean

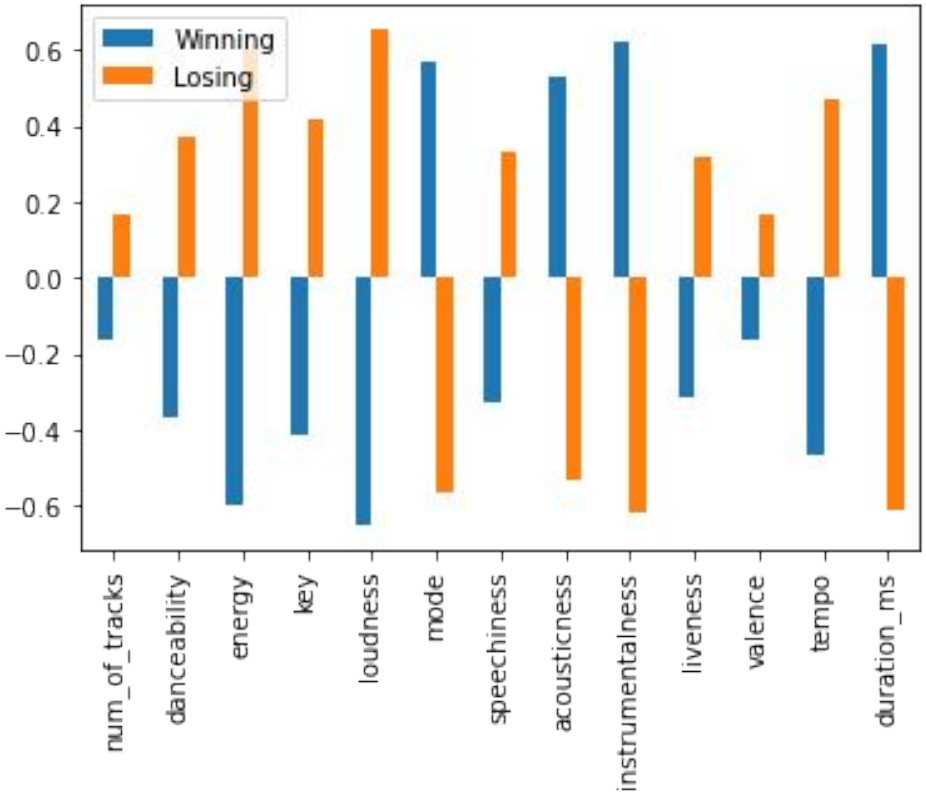
0.63

Decision trees

“Gini”, “random”
MinMaxScaler

Prediction

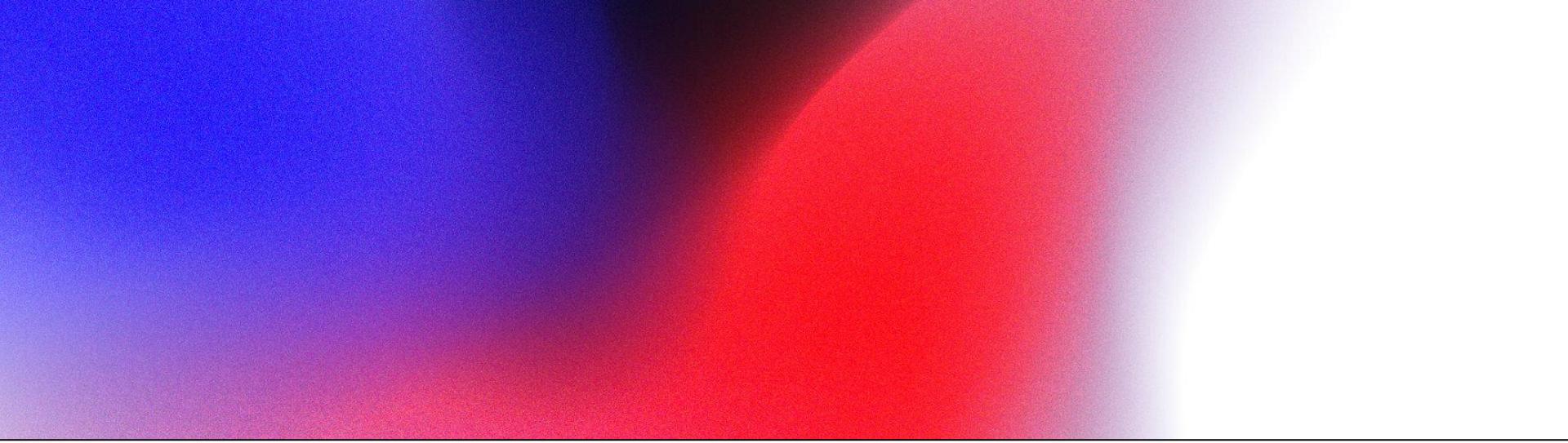
Prediction Probabilities



Conclusion

Outcomes:

- We ran our model with a testing dataset of albums currently released in 2023 (really early in the musical year):
  - The winning songs were those with a lower average speechiness, lower energy, and higher acoustic-ness.
  - Confirmed by correlation values plotted previously.
- What does this mean?
  - Biased against music that is more lyrical and energetic
  - Biased towards music that is more acoustic
- Final Conclusion:
  - We conclude that the Grammy's are biased towards genres who oftentimes lack the characteristics described above like rap, hip-hop, and R&B.



THANK YOU