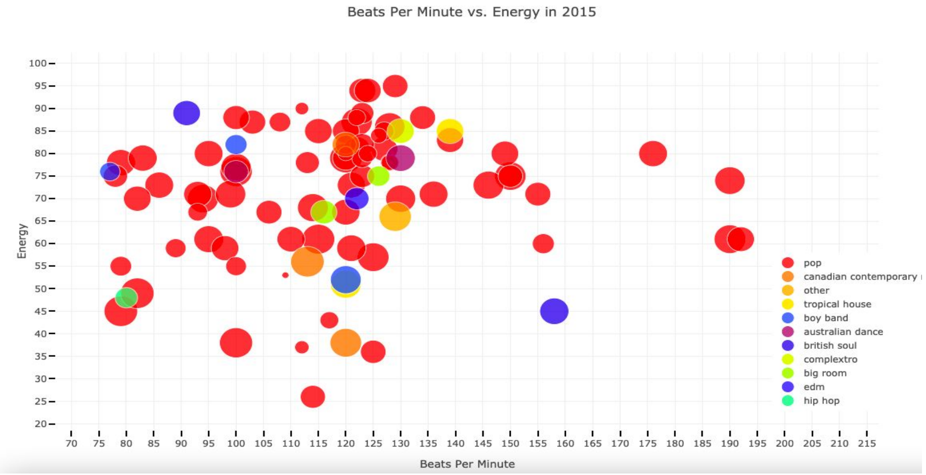
Billboard Song Prediction Project

Teammates: Sarah D. Hood, Jini Hassan, Ryan Eccleston-Murdock, Wasif Khan, Angeli Lucila, Ivana Korak

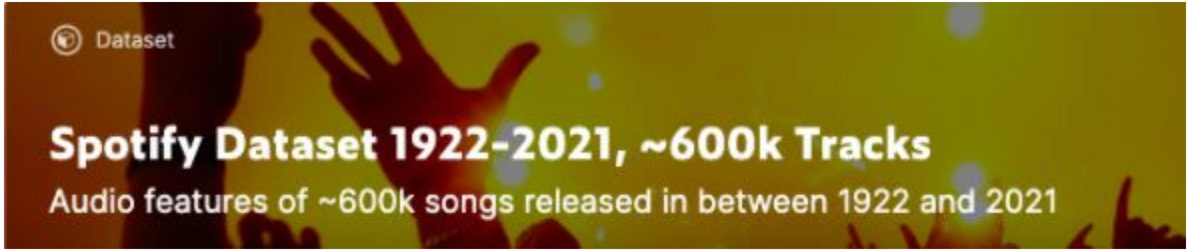
Inspiration and Goal

- Building on past project
  - Rise of British Soul - could we predict?
- Try to predict song attributes/genres that will be in the next hit songs
- Models:
  - Time Series?
  - Logarithmic
  - SVM
- Plan:
  - Billboard weekly top 100 songs
  - Predict next week’s hits



Datasets

- CSV: Spotify songs with attributes 1922-2021
- CSV: Billboard weekly top 100 1958-2020
- Scraping: Billboard weekly top 100 2021
- Calls to Spotify API for unique Spotify IDs and/or song attributes



Dataset

Spotify Dataset 1922-2021, ~600k Tracks

Audio features of ~600k songs released in between 1922 and 2021

Billboard Hot weekly charts

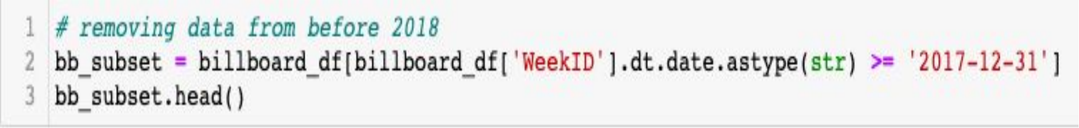
DATASET BY SEAN MILLER

Workflow Overview

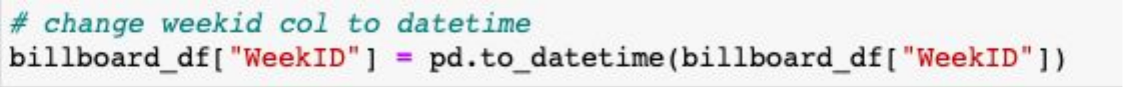
- Clean existing CSVs
- Scraping Billboard
- Make Spotify API calls
- Join CSVs on Spotify ID
- Prepare the data for modelling
- Run the models

CSV cleaning: code

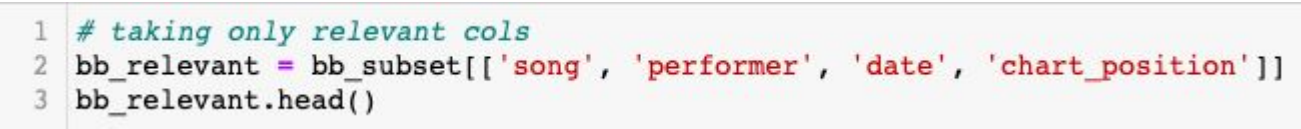
- Cutting to 2018 - 2020
- Data types: datetime
- Cleaning strings
- Filling NaNs
- Removing unnecessary columns



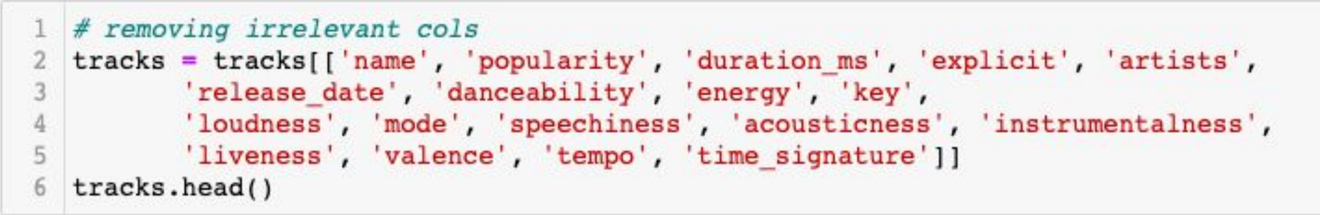
```
# removing data from before 2018
bb_subset = billboard_df[billboard_df['WeekID'].dt.date.astype(str) >= '2017-12-31']
bb_subset.head()
```



```
# change weekid col to datetime
billboard_df["WeekID"] = pd.to_datetime(billboard_df["WeekID"])
```

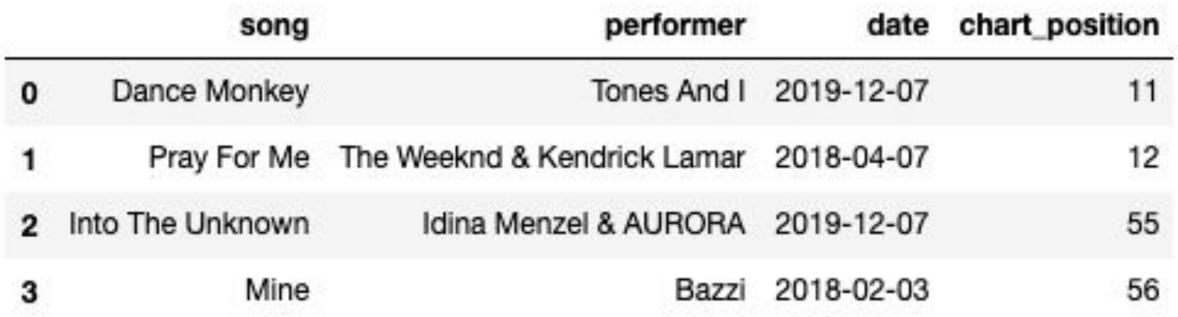


```
# taking only relevant cols
bb_relevant = bb_subset[['song', 'performer', 'date', 'chart_position']]
bb_relevant.head()
```

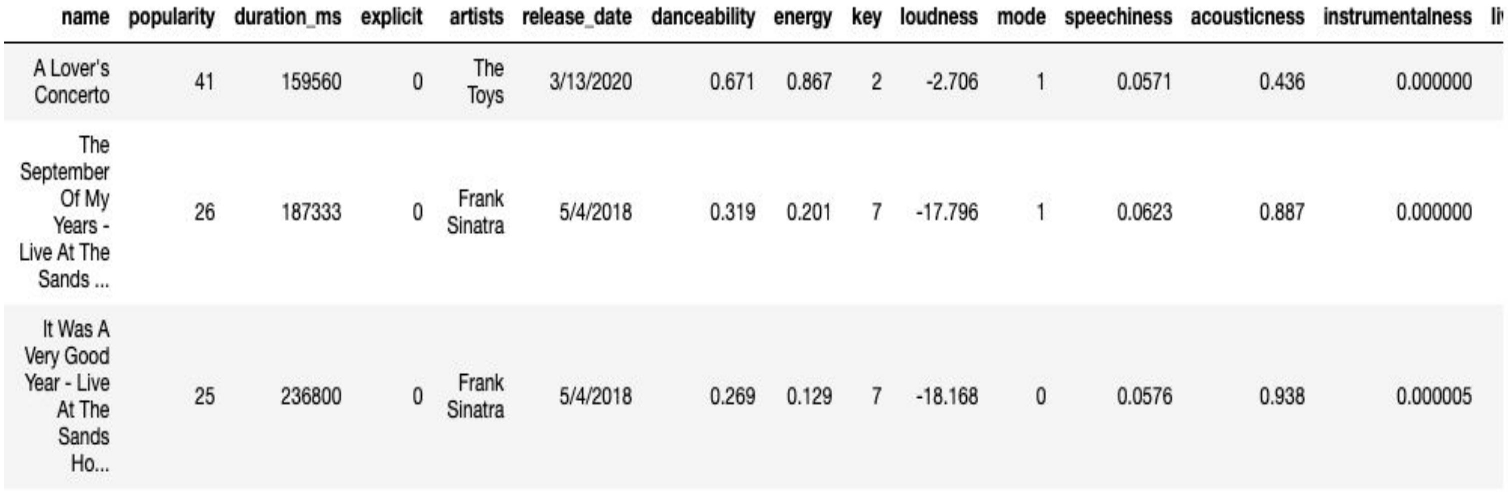


```
# removing irrelevant cols
tracks = tracks[['name', 'popularity', 'duration_ms', 'explicit', 'artists',
       'release_date', 'danceability', 'energy', 'key',
       'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness',
       'liveness', 'valence', 'tempo', 'time_signature']]
tracks.head()
```

CSV cleaning: Results

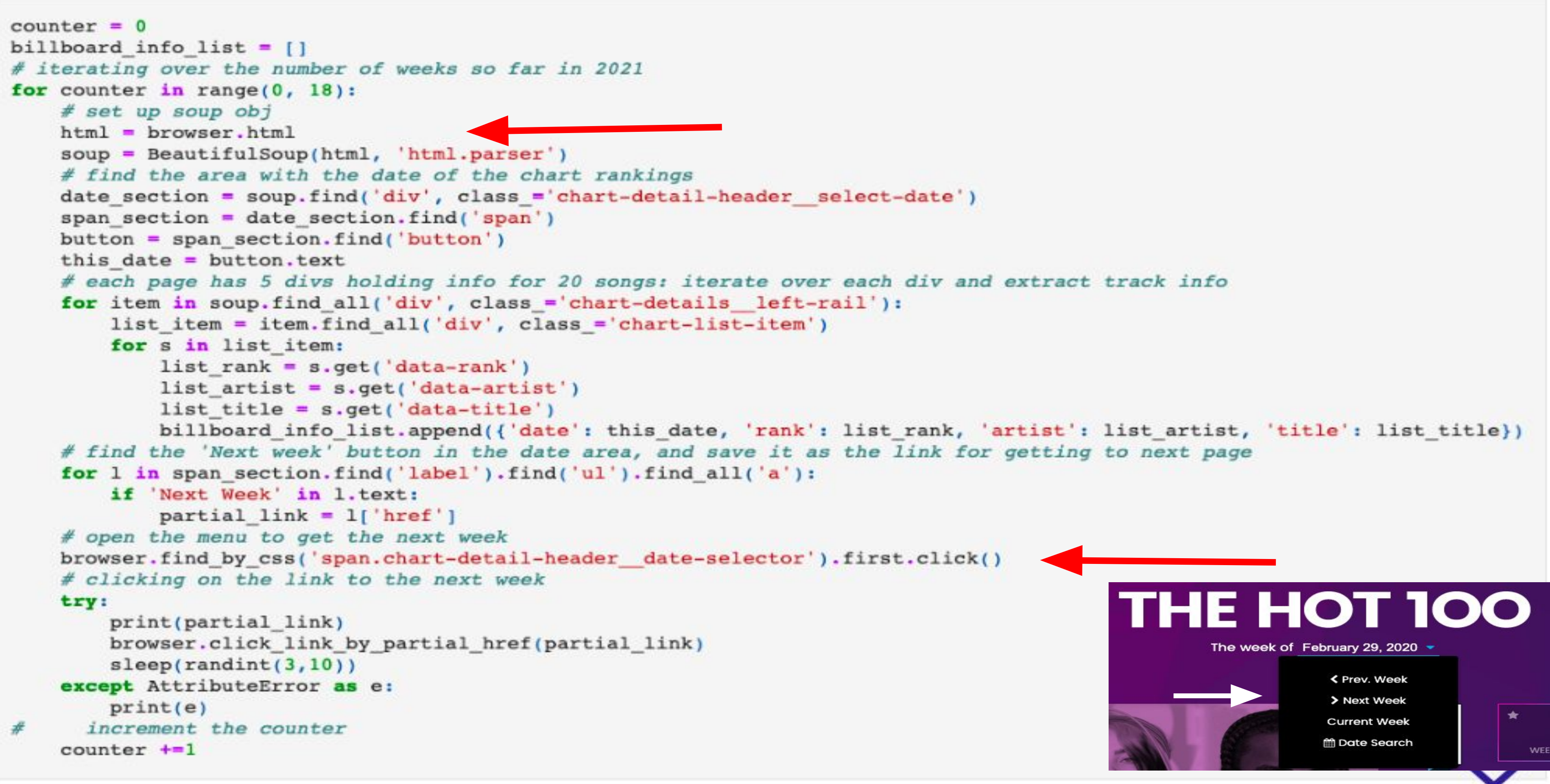


	song	performer	date	chart_position
0	Dance Monkey	Tones And I	2019-12-07	11
1	Pray For Me	The Weeknd & Kendrick Lamar	2018-04-07	12
2	Into The Unknown	Idina Menzel & AURORA	2019-12-07	55
3	Mine	Bazzi	2018-02-03	56



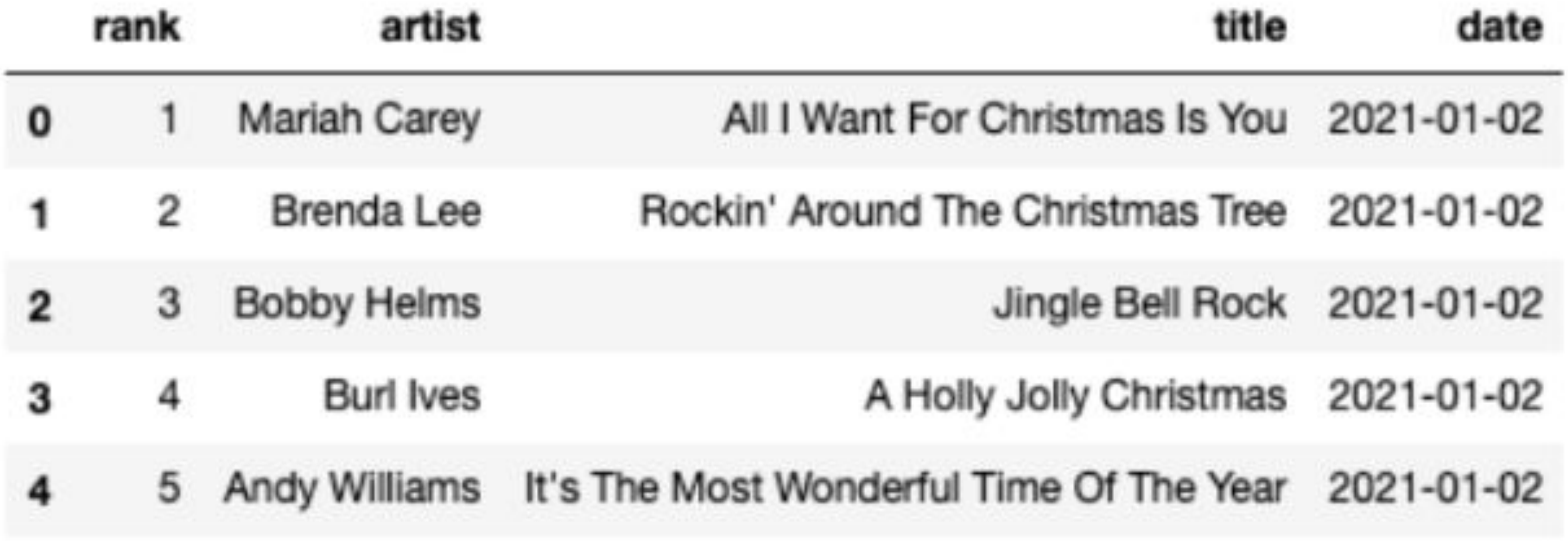
name	popularity	duration_ms	explicit	artists	release_date	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	li
A Lover's Concerto	41	159560	0	The Toys	3/13/2020	0.671	0.867	2	-2.706	1	0.0571	0.436	0.000000	
The September Of My Years - Live At The Sands ...	26	187333	0	Frank Sinatra	5/4/2018	0.319	0.201	7	-17.796	1	0.0623	0.887	0.000000	
It Was A Very Good Year - Live At The Sands Ho...	25	236800	0	Frank Sinatra	5/4/2018	0.269	0.129	7	-18.168	0	0.0576	0.938	0.000005	

Billboard Scraping: Code & Challenges



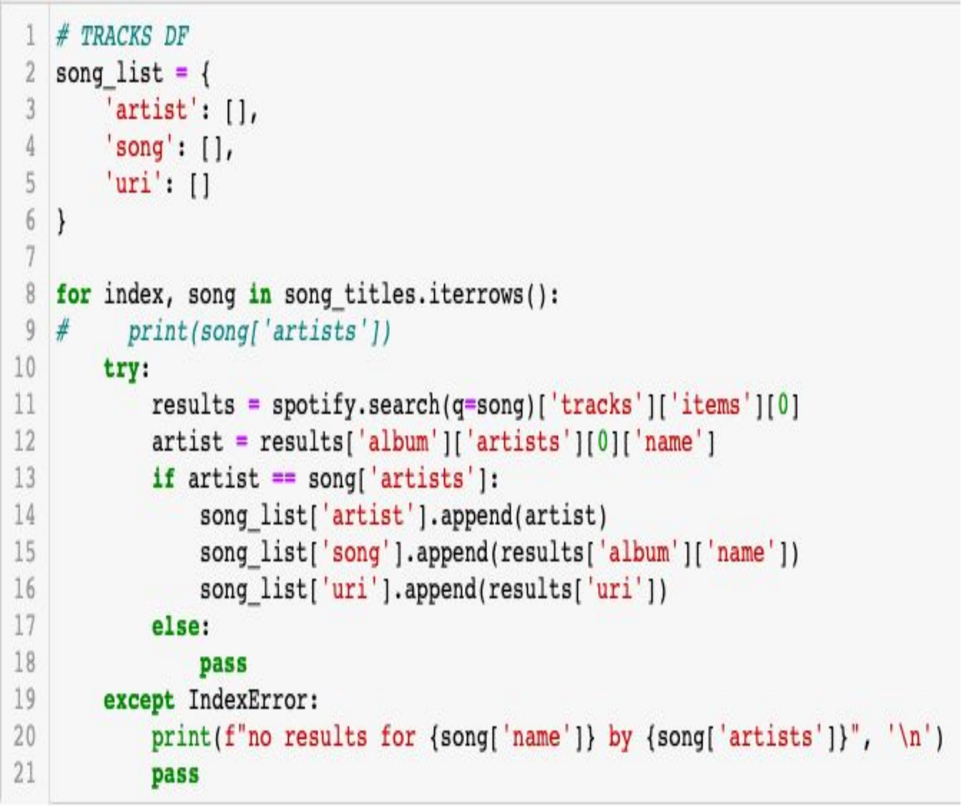
```
counter = 0
billboard_info_list = []
# iterating over the number of weeks so far in 2021
for counter in range(0, 18):
    # set up soup obj
    html = browser.html
    soup = BeautifulSoup(html, 'html.parser')
    # find the area with the date of the chart rankings
    date_section = soup.find('div', class_='chart-detail-header__select-date')
    span_section = date_section.find('span')
    button = span_section.find('button')
    this_date = button.text
    # each page has 5 divs holding info for 20 songs: iterate over each div and extract track info
    for item in soup.find_all('div', class_='chart-details__left-rail'):
        list_item = item.find_all('div', class_='chart-list-item')
        for s in list_item:
            list_rank = s.get('data-rank')
            list_artist = s.get('data-artist')
            list_title = s.get('data-title')
            billboard_info_list.append({'date': this_date, 'rank': list_rank, 'artist': list_artist, 'title': list_title})
    # find the 'Next week' button in the date area, and save it as the link for getting to next page
    for l in span_section.find('label').find('ul').find_all('a'):
        if 'Next Week' in l.text:
            partial_link = l['href']
    # open the menu to get the next week
    browser.find_by_css('span.chart-detail-header__date-selector').first.click()
    # clicking on the link to the next week
    try:
        print(partial_link)
        browser.click_link_by_partial_href(partial_link)
        sleep(randint(3,10))
    except AttributeError as e:
        print(e)
#     increment the counter
    counter +=1
```

Billboard Scraping: Results

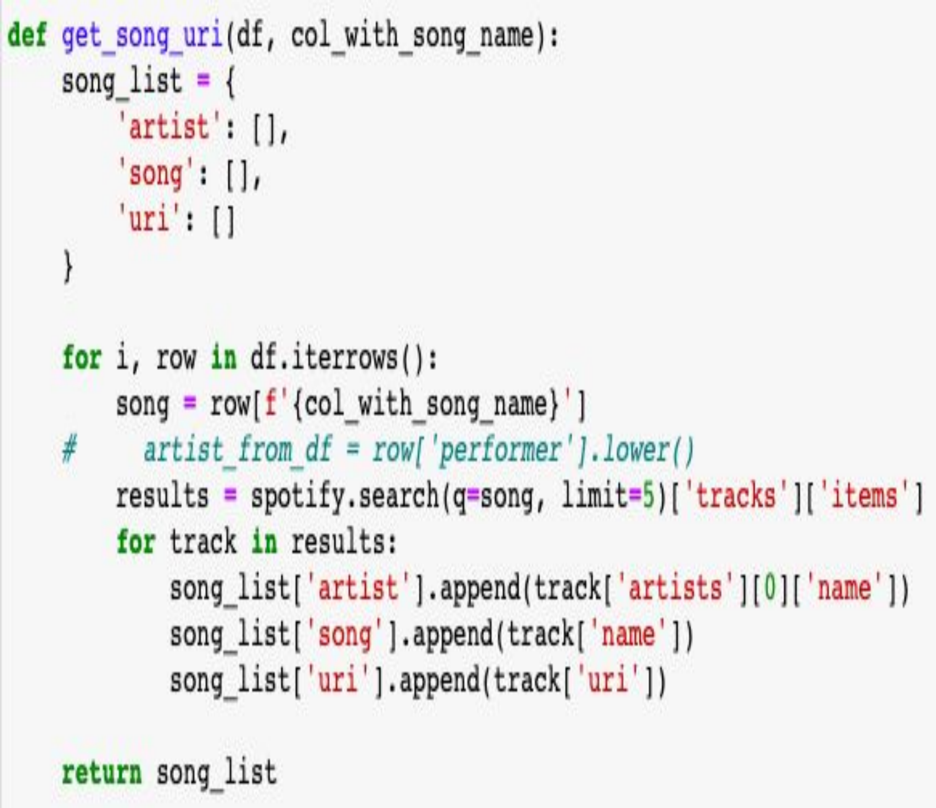


	rank	artist	title	date
0	1	Mariah Carey	All I Want For Christmas Is You	2021-01-02
1	2	Brenda Lee	Rockin' Around The Christmas Tree	2021-01-02
2	3	Bobby Helms	Jingle Bell Rock	2021-01-02
3	4	Burl Ives	A Holly Jolly Christmas	2021-01-02
4	5	Andy Williams	It's The Most Wonderful Time Of The Year	2021-01-02

Spotify API calls



```
# TRACKS DF
song_list = {
    'artist': [],
    'song': [],
    'uri': []
}

for index, song in song_titles.iterrows():
#     print(song['artists'])
    try:
        results = spotify.search(q=song)['tracks']['items'][0]
        artist = results['album']['artists'][0]['name']
        if artist == song['artists']:
            song_list['artist'].append(artist)
            song_list['song'].append(results['album']['name'])
            song_list['uri'].append(results['uri'])
        else:
            pass
    except IndexError:
        print(f"no results for {song['name']} by {song['artists']}", '\n')
        pass
```



```
def get_song_uri(df, col_with_song_name):
    song_list = {
        'artist': [],
        'song': [],
        'uri': []
    }

    for i, row in df.iterrows():
        song = row[f'{col_with_song_name}']
#       artist_from_df = row['performer'].lower()
        results = spotify.search(q=song, limit=5)['tracks']['items']
        for track in results:
            song_list['artist'].append(track['artists'][0]['name'])
            song_list['song'].append(track['name'])
            song_list['uri'].append(track['uri'])

    return song_list
```

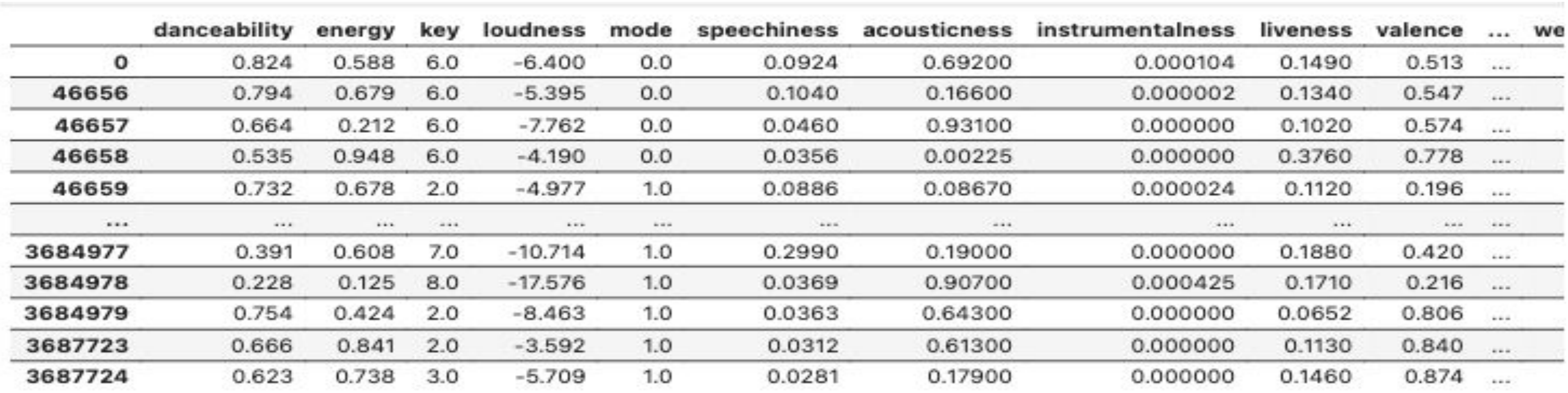
Spotify API calls: Challenges

- Time
- Results per track

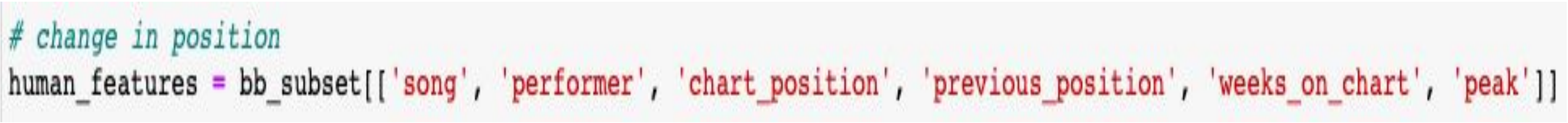
Change of Direction!

Problems with our current inputs...

- Preliminary models: song attributes don't have much predictive ability
- Better factors
- Time limit: only 2018 - 2020 data

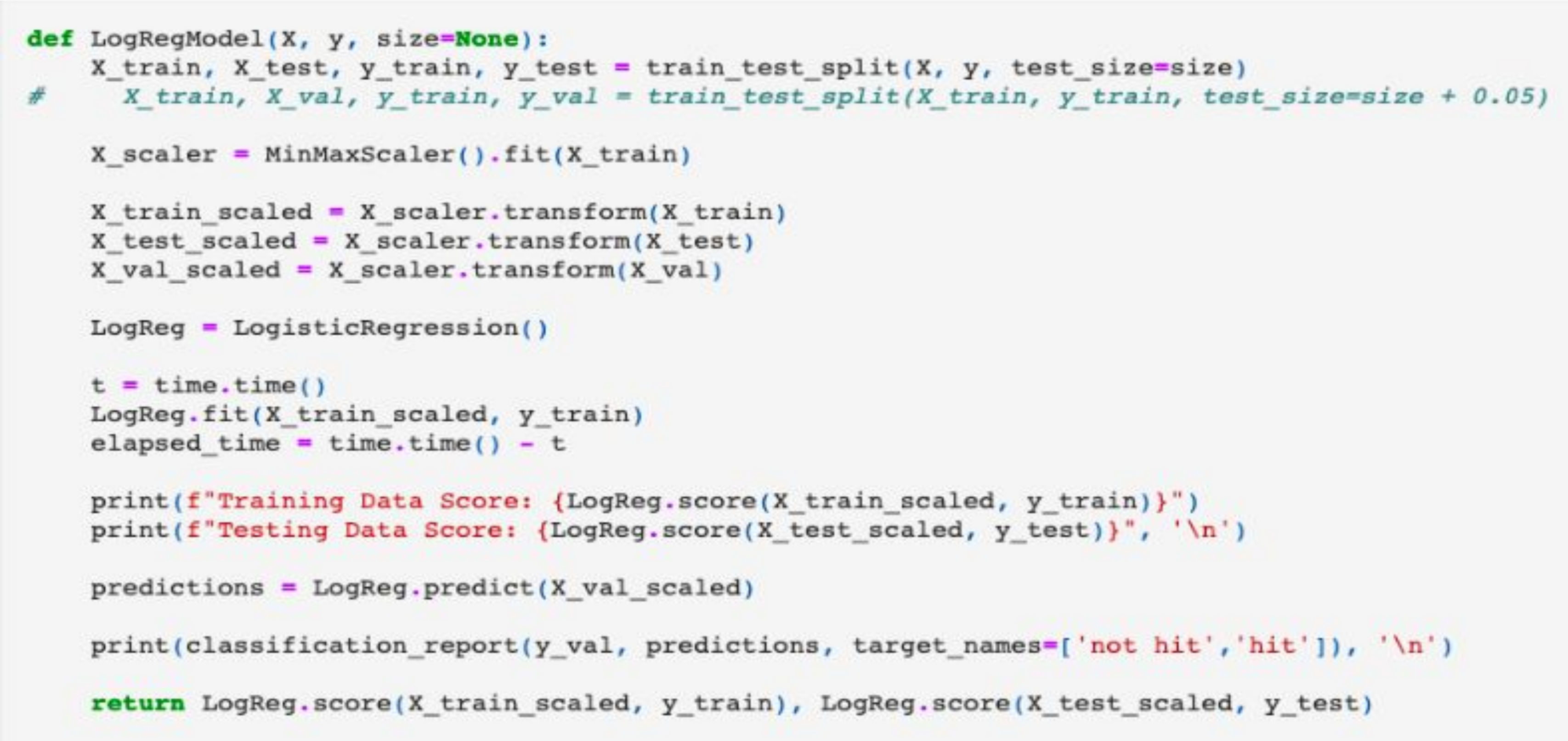


	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	...	we
0	0.824	0.588	6.0	-6.400	0.0	0.0924	0.69200	0.000104	0.1490	0.513	...	
46656	0.794	0.679	6.0	-5.395	0.0	0.1040	0.16600	0.000002	0.1340	0.547	...	
46657	0.664	0.212	6.0	-7.762	0.0	0.0460	0.93100	0.000000	0.1020	0.574	...	
46658	0.535	0.948	6.0	-4.190	0.0	0.0356	0.00225	0.000000	0.3760	0.778	...	
46659	0.732	0.678	2.0	-4.977	1.0	0.0886	0.08670	0.000024	0.1120	0.196	...	
...	...	...	...	...	...	...	...	...	...	...	...	
3684977	0.391	0.608	7.0	-10.714	1.0	0.2990	0.19000	0.000000	0.1880	0.420	...	
3684978	0.228	0.125	8.0	-17.576	1.0	0.0369	0.90700	0.000425	0.1710	0.216	...	
3684979	0.754	0.424	2.0	-8.463	1.0	0.0363	0.64300	0.000000	0.0652	0.806	...	
3687723	0.666	0.841	2.0	-3.592	1.0	0.0312	0.61300	0.000000	0.1130	0.840	...	
3687724	0.623	0.738	3.0	-5.709	1.0	0.0281	0.17900	0.000000	0.1460	0.874	...	



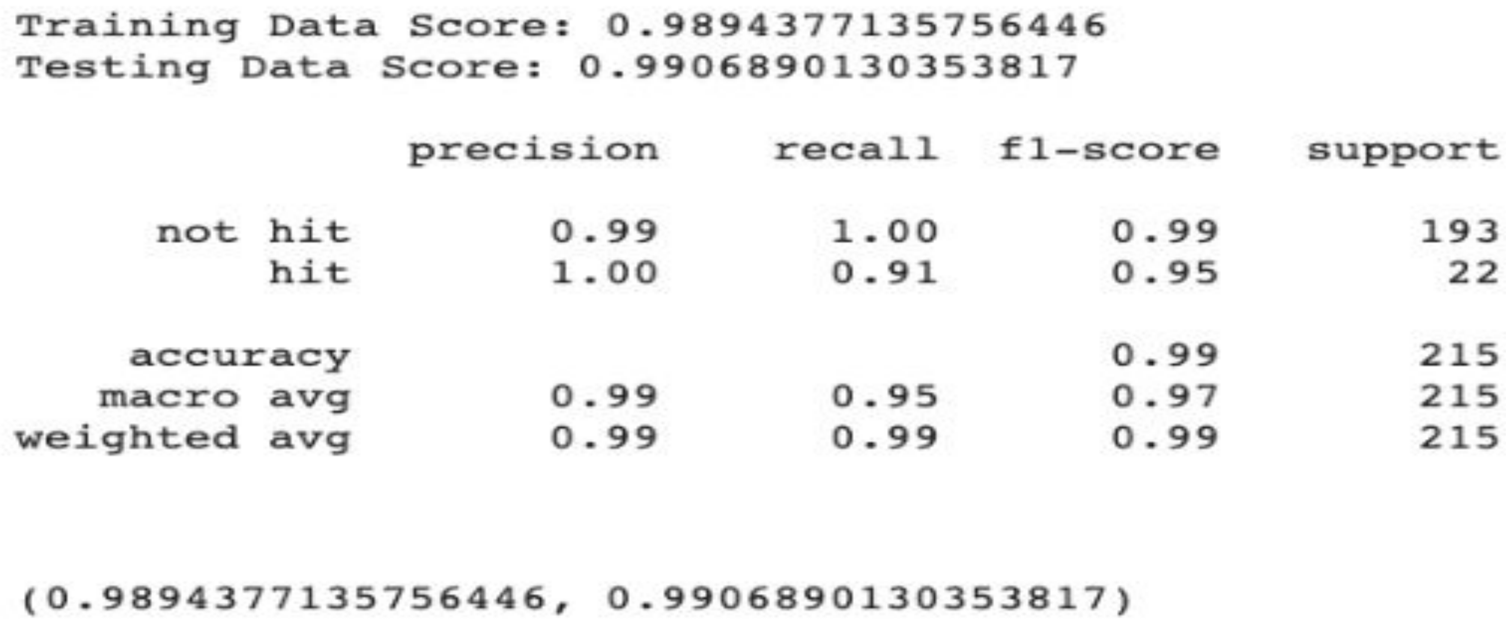
```
# change in position
human_features = bb_subset[['song', 'performer', 'chart_position', 'previous_position', 'weeks_on_chart', 'peak']]
```

Logarithmic Model: Code



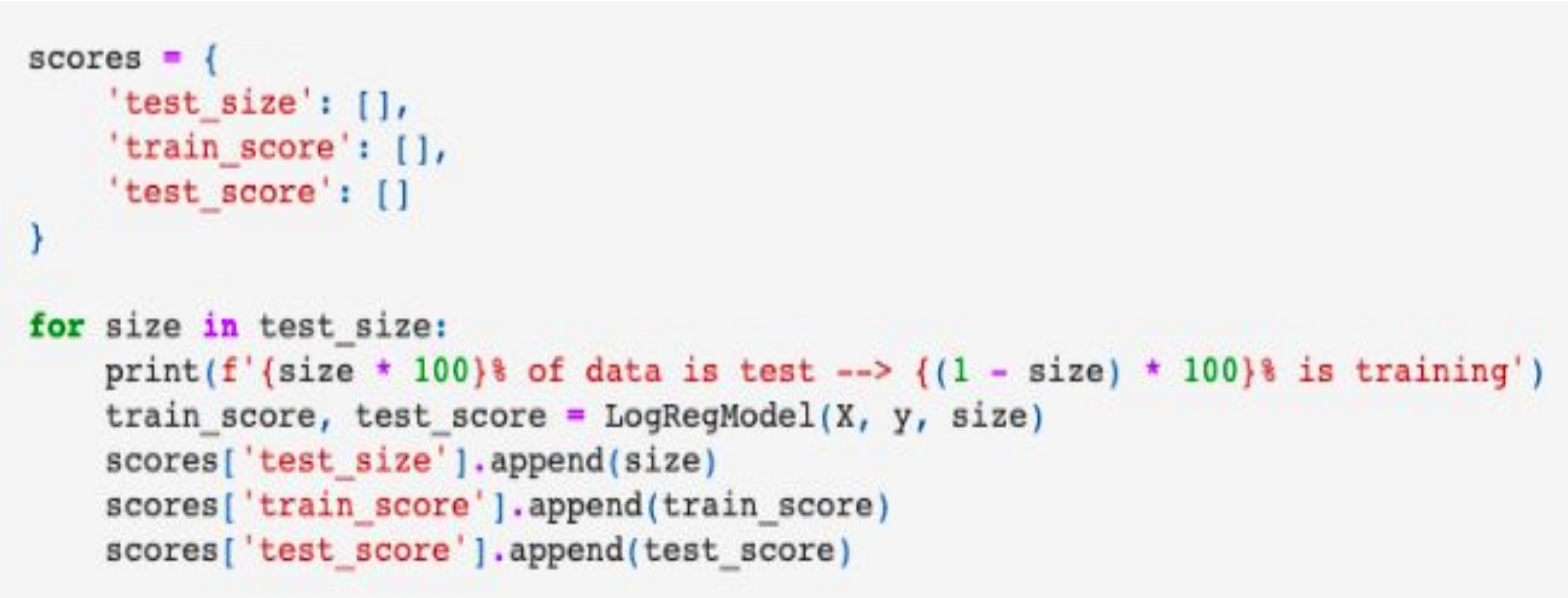
```
def LogRegModel(X, y, size=None):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=size)
#     X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=size + 0.05)

    X_scaler = MinMaxScaler().fit(X_train)

    X_train_scaled = X_scaler.transform(X_train)
    X_test_scaled = X_scaler.transform(X_test)
    X_val_scaled = X_scaler.transform(X_val)

    LogReg = LogisticRegression()

    t = time.time()
    LogReg.fit(X_train_scaled, y_train)
    elapsed_time = time.time() - t

    print(f"Training Data Score: {LogReg.score(X_train_scaled, y_train)}")
    print(f"Testing Data Score: {LogReg.score(X_test_scaled, y_test)}", '\n')

    predictions = LogReg.predict(X_val_scaled)

    print(classification_report(y_val, predictions, target_names=['not hit','hit']), '\n')

    return LogReg.score(X_train_scaled, y_train), LogReg.score(X_test_scaled, y_test)
```

Logarithmic Model: Results

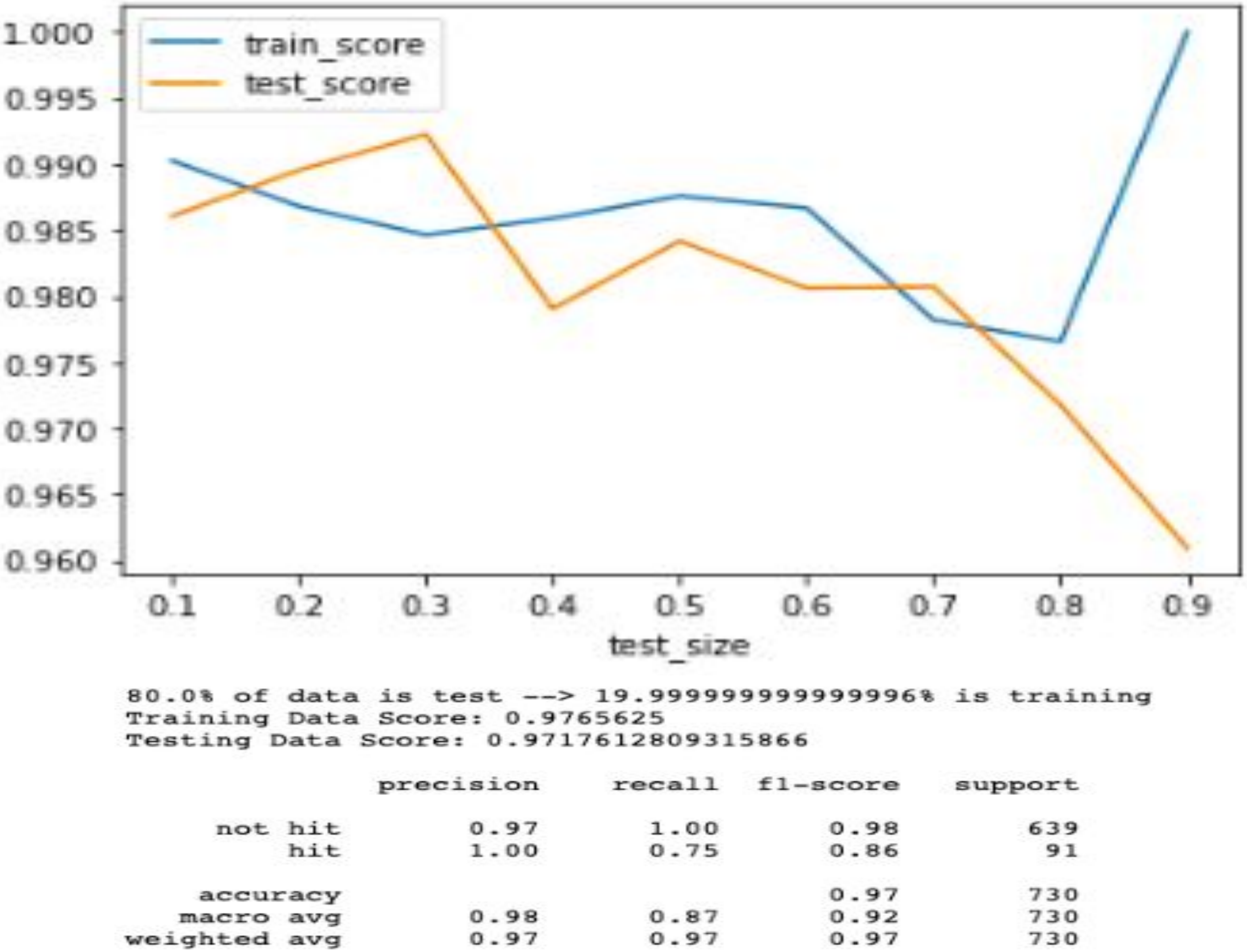


```
Training Data Score: 0.9894377135756446
Testing Data Score: 0.9906890130353817

              precision    recall  f1-score   support

     not hit       0.99      1.00      0.99       193
         hit       1.00      0.91      0.95        22

    accuracy                           0.99       215
   macro avg       0.99      0.95      0.97       215
weighted avg       0.99      0.99      0.99       215



(0.9894377135756446, 0.9906890130353817)
```

Modulating Test and Training Percentages

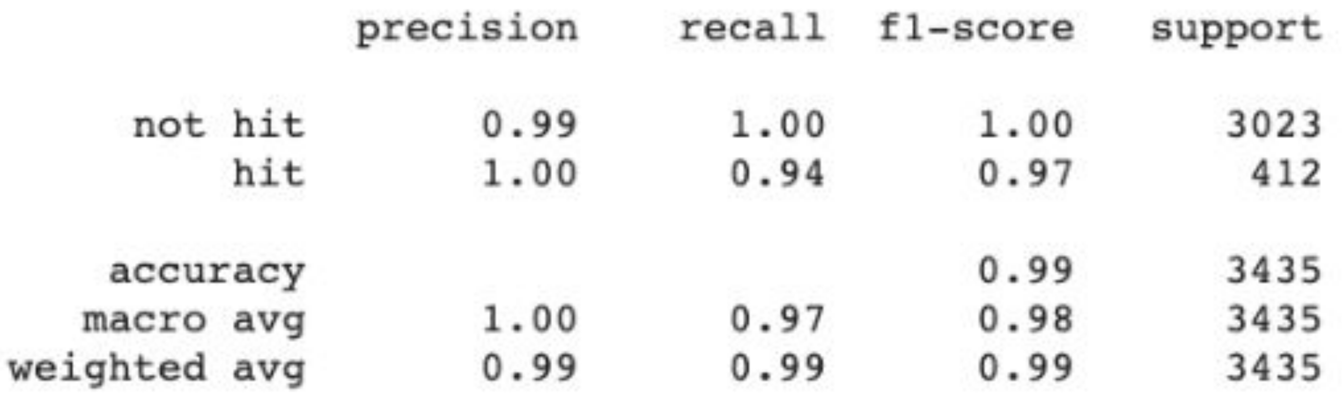


```
scores = {
    'test_size': [],
    'train_score': [],
    'test_score': []
}

for size in test_size:
    print(f'{size * 100}% of data is test --> {(1 - size) * 100}% is training')
    train_score, test_score = LogRegModel(X, y, size)
    scores['test_size'].append(size)
    scores['train_score'].append(train_score)
    scores['test_score'].append(test_score)
```



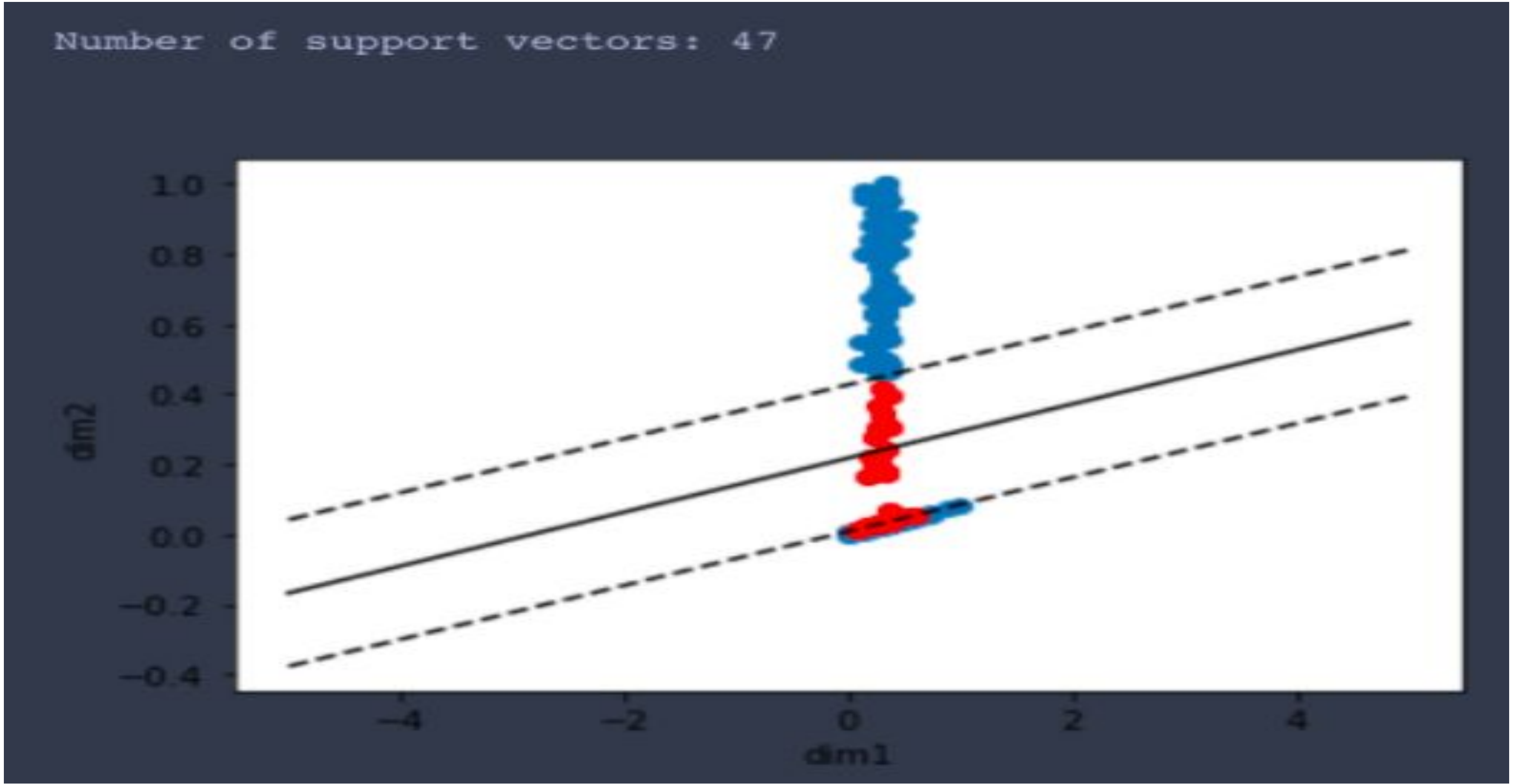
```
80.0% of data is test --> 19.999999999999996% is training
Training Data Score: 0.9765625
Testing Data Score: 0.9717612809315866

              precision    recall  f1-score   support

     not hit       0.97      1.00      0.98       639
         hit       1.00      0.75      0.86        91

    accuracy                           0.97       730
   macro avg       0.98      0.87      0.92       730
weighted avg       0.97      0.97      0.97       730
```

SVM Model



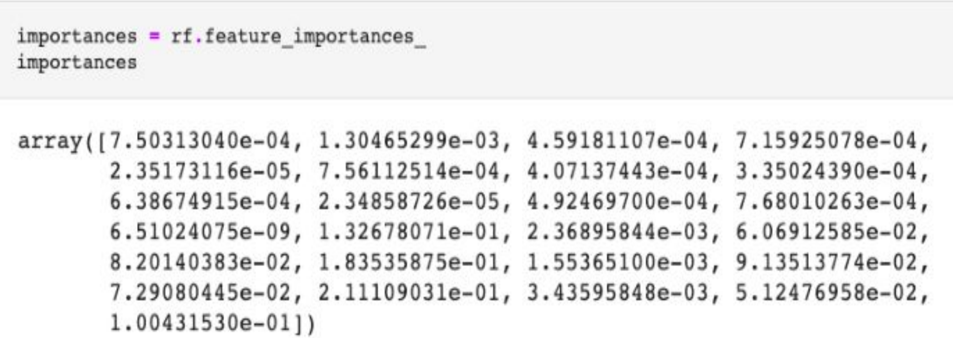
```
              precision    recall  f1-score   support

     not hit       0.99      1.00      1.00      3023
         hit       1.00      0.94      0.97       412

    accuracy                           0.99      3435
   macro avg       1.00      0.97      0.98      3435
weighted avg       0.99      0.99      0.99      3435
```

SVM: Graphed



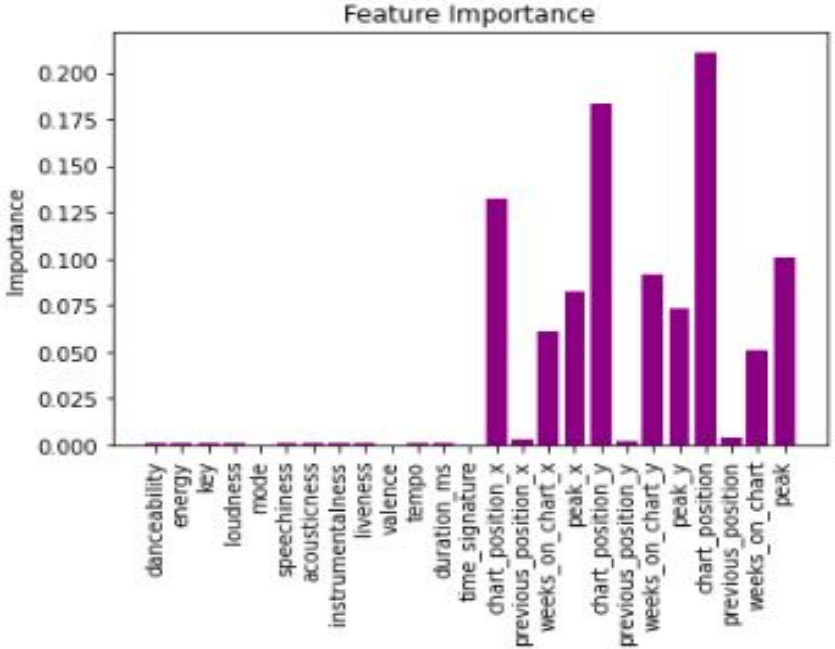
```
Number of support vectors: 47
```

Random Forest: Scoring the Model



```
importances = rf.feature_importances_
importances
```

```
array([7.50313040e-04, 1.30465299e-03, 4.59181107e-04, 7.15925078e-04,
       2.35173116e-05, 7.56112514e-04, 4.07137443e-04, 3.35024390e-04,
       6.38674915e-04, 2.34858726e-05, 4.92469700e-04, 7.68010263e-04,
       6.51024075e-09, 1.32678071e-01, 2.36895844e-03, 6.06912585e-02,
       8.20140383e-02, 1.83535875e-01, 1.55365100e-03, 9.13513774e-02,
       7.29080445e-02, 2.11109031e-01, 3.43595848e-03, 5.12476958e-02,
       1.00431530e-01])
```



Future Directions/Do-Overs

- Getting advice about type of model earlier
- Not clipping the 2018-2020 dataset to take away the weeks on the chart and top ranking columns
- Add in the 2021 data to make it up to date
- Make the model live on a website, with data in a database, updated every week

Thank you!

- Model advice
- Repo example
- Help with scraping

Questions?