

CA1_spotify

February 15, 2024

1 CA1: Dataframe Manipulation with Spotify Data

1.1 Introduction

Pandas is an extremely powerful tool to handle large amounts of tabular data. In this compulsory assignment, you will use Pandas to explore one of the TA's personal spotify data in depth.

Additional information: - Feel free to create additional code cells if you feel that one cell per subtask is not sufficient. - Remember, Pandas uses very efficient code to handle large amounts of data. For-loops are not efficient. If you ever have to use a for-loop to loop over the rows in the DataFrame, you have *probably* done something wrong. - Label all graphs and charts if applicable.

1.2 Task

I typically enjoy indie and rock music. I am a big fan of everything from old-fashioned rock and roll like Led Zeppelin and Jimi Hendrix, to newer indie artists like Joji and Lana Del Rey. This is why my spotify wrapped for 2023 came as quite a surprise:

Now, I'm no hater of pop music, but this was unexpected. For this assignment, you will investigate my listening habits, including a deep dive into my Ariana Grande listening habits, and try to find an answer to why she was my top artist; was there a fault in the spotify algorithm? Am I actually secretly an *Arianator*? (yes, I did have to look that up). Or am I just lying to myself about how often I listen to guilty pleasure music?

1.3 Part 1: Initial loading and exploration

1.0 Import necessary libraries: pandas, numpy, matplotlib.pyplot (other libraries such as seaborn or plotly are also allowed if you want prettier plots). It might also be a good idea to use `os` for task 2.0

```
[235]: import matplotlib.pyplot as plt
import pandas as pd
import os
```

1.1 Loading the data Load the dataset in the file `streaming_history_0.csv` into a Pandas DataFrame called `df_spotify_0`.

```
[236]: df_spotify_0 = pd.read_csv('spotify_data/streaminghistory0.csv')
```

1.2 Help function Use the Python command `help` to help you understand how to use the `pd.DataFrame.head` and `pd.DataFrame.tail` methods.

```
[237]: help(pd.DataFrame.head)
```

Help on function head in module pandas.core.generic:

```
head(self, n: 'int' = 5) -> 'Self'
    Return the first `n` rows.
```

This function returns the first `n` rows for the object based on position. It is useful for quickly testing if your object has the right type of data in it.

For negative values of `n`, this function returns all rows except the last `|n|` rows, equivalent to `df[:n]`.

If `n` is larger than the number of rows, this function returns all rows.

Parameters

`n` : int, default 5
 Number of rows to select.

Returns

same type as caller
 The first `n` rows of the caller object.

See Also

`DataFrame.tail`: Returns the last `n` rows.

Examples

```
>>> df = pd.DataFrame({'animal': ['alligator', 'bee', 'falcon', 'lion',
...                               'monkey', 'parrot', 'shark', 'whale', 'zebra']})
```

```
>>> df
   animal
0  alligator
1      bee
2   falcon
3     lion
4   monkey
5   parrot
6    shark
7    whale
8    zebra
```

Viewing the first 5 lines

```
>>> df.head()
      animal
0  alligator
1        bee
2      falcon
3        lion
4      monkey
```

Viewing the first `n` lines (three in this case)

```
>>> df.head(3)
      animal
0  alligator
1        bee
2      falcon
```

For negative values of `n`

```
>>> df.head(-3)
      animal
0  alligator
1        bee
2      falcon
3        lion
4      monkey
5      parrot
```

[238]: `help(pd.DataFrame.tail)`

Help on function tail in module pandas.core.generic:

```
tail(self, n: 'int' = 5) -> 'Self'
    Return the last `n` rows.
```

This function returns last `n` rows from the object based on position. It is useful for quickly verifying data, for example, after sorting or appending rows.

For negative values of `n`, this function returns all rows except the first `|n|` rows, equivalent to `df[|n|:]`.

If `n` is larger than the number of rows, this function returns all rows.

Parameters

`n : int, default 5`
Number of rows to select.

Returns

type of caller
The last ``n`` rows of the caller object.

See Also

`DataFrame.head` : The first ``n`` rows of the caller object.

Examples

```
>>> df = pd.DataFrame({'animal': ['alligator', 'bee', 'falcon', 'lion',  
...                               'monkey', 'parrot', 'shark', 'whale', 'zebra']})
```

```
>>> df
      animal
0  alligator
1         bee
2      falcon
3         lion
4      monkey
5      parrot
6       shark
7       whale
8       zebra
```

Viewing the last 5 lines

```
>>> df.tail()
      animal
4  monkey
5  parrot
6   shark
7   whale
8   zebra
```

Viewing the last ``n`` lines (three in this case)

```
>>> df.tail(3)
      animal
6   shark
7   whale
8   zebra
```

For negative values of ``n``

```
>>> df.tail(-3)
      animal
3      lion
4  monkey
5  parrot
6   shark
7   whale
8   zebra
```

1.3 Getting an overview Print the first five and last ten rows of the dataframe. Have a quick look at which columns are in the dataset.

```
[239]: df_spotify_0.head()
```

```
[239]:
```

	endTime	artistName	trackName \
0	2022-12-03 02:02	Cigarettes After Sex	Truly
1	2022-12-03 02:02	Leonard Cohen	Take This Waltz - Paris Version
2	2022-12-06 21:05	Vlad Holiday	So Damn Into You
3	2022-12-06 21:05	Lorde	Team
4	2022-12-06 21:05	Ariana Grande	Into You

	msPlayed
0	30000.0
1	8210.0
2	37895.0
3	8984.0
4	1221.0

```
[240]: df_spotify_0.tail(10)
```

```
[240]:
```

	endTime	artistName	trackName \
11949	2023-01-02 20:58	Ariana Grande	six thirty
11950	2023-01-02 20:58	Leonard Cohen	Thanks for the Dance
11951	2023-01-02 20:59	Des Rocs	Used to the Darkness
11952	2023-01-02 20:59	Caroline Polachek	Hit Me Where It Hurts
11953	2023-01-02 20:59	Caroline Polachek	Hit Me Where It Hurts
11954	2023-01-02 20:59	Kaizers Orchestra	Resistansen
11955	2023-01-02 20:59	Mr.Kitty	After Dark
11956	2023-01-02 20:59	daddy's girl	after dark x sweater weather
11957	2023-01-02 20:59	daddy's girl	after dark x sweater weather
11958	2023-01-02 20:59	daddy's girl	after dark x sweater weather

	msPlayed
11949	1699.0
11950	19483.0
11951	185.0

```

11952    603.0
11953    208.0
11954    208.0
11955  101447.0
11956    301.0
11957    208.0
11958    789.0

```

1.4 Formatting correctly When working with Pandas, it's very useful to have columns which contains dates in a specific format called *datetime*. This allows for efficient manipulation and analysis of time-series data, such as sorting, filtering by date or time, and resampling for different time periods. Figure out which column(s) would be appropriate to convert to datetime, if any, and if so, perform the conversion to the correct format.

```
[241]: df_spotify_0['endTime'] = pd.to_datetime(df_spotify_0['endTime'])
df_spotify_0.head()
```

```
[241]:
```

	endTime	artistName	trackName \
0	2022-12-03 02:02:00	Cigarettes After Sex	Truly
1	2022-12-03 02:02:00	Leonard Cohen	Take This Waltz - Paris Version
2	2022-12-06 21:05:00	Vlad Holiday	So Damn Into You
3	2022-12-06 21:05:00	Lorde	Team
4	2022-12-06 21:05:00	Ariana Grande	Into You


```

msPlayed
0    30000.0
1     8210.0
2    37895.0
3     8984.0
4     1221.0

```

1.5 Unique artists Find how many unique artists are in the dataset.

```
[242]: df_spotify_0['artistName'].nunique()
```

```
[242]: 495
```

1.6 Unique songs Find how many unique songs are in the dataset.

```
[243]: df_spotify_0['trackName'].nunique()
```

```
[243]: 1308
```

1.3.1 Part 1: Questions

Q1: Which columns are in the dataset? endTime, artistName, trackName, msPlayed

Q2: What timeframe does the dataset span? 2022-12-03 02:02 to 2023-01-02 20:59:00

Q3: How many unique artists are in the dataset? 495

Q4: How many unique songs are in the dataset? 1308

1.4 Part 2: Working with all the data

2.0 Importing all the dataframes In Task 1, you only worked with about a month worth of data. Now, you will work with over a year worth.

In the *spotify_data* folder, there is more than just one listening record. Load each of the 14 listening records into a dataframe (1 dataframe per listening record), and concatenate them together into one large dataframe named *df*.

```
[244]: data_dir = 'spotify_data'
dfs = []

for file_name in os.listdir(data_dir):
    df_temp = pd.read_csv(os.path.join(data_dir, file_name))
    dfs.append(df_temp)
df = pd.concat(dfs, ignore_index=True)
```

2.1 Sorting by time Datasets often aren't perfect. One example of an issue that could occur is that the time-based data might not be in chronological order. If this were to happen, the rows in your dataframe could be in the wrong order. To ensure this isn't an issue in your dataframe, you should sort the dataframe in chronological order, from oldest to newest.

```
[245]: df['endTime'] = pd.to_datetime(df['endTime'])
df = df.sort_values(by='endTime')
print(df)
```

	endTime	artistName \
71767	2022-12-03 02:02:00	Cigarettes After Sex
71768	2022-12-03 02:02:00	Leonard Cohen
71770	2022-12-06 21:05:00	Lorde
71769	2022-12-06 21:05:00	Vlad Holiday
71771	2022-12-06 21:05:00	Ariana Grande
...
59802	2023-12-07 21:13:00	Childish Gambino
59806	2023-12-07 21:14:00	Leonard Cohen
59805	2023-12-07 21:14:00	Ariana Grande
59807	2023-12-07 21:17:00	The Vaccines
56898	NaT	The Lumineers

	trackName	msPlayed
71767	Truly	30000.0
71768	Take This Waltz - Paris Version	8210.0
71770	Team	8984.0
71769	So Damn Into You	37895.0
71771	Into You	1221.0

```

...
59802                Les    2484.0
59806      Thanks for the Dance    9317.0
59805                my hair    23757.0
59807  Your Love Is My Favourite Band    14661.0
56898                Ophelia     371.0

```

[167439 rows x 4 columns]

2.2 Setting a timeframe For this investigation, we are only interested in investigating listening patterns from **2023**. Remove any data not from **2023** from the DataFrame.

```
[246]: df_2023 = df[df['endTime'].dt.year == 2023]
df_2023
```

```
[246]:
```

	endTime	artistName	trackName \
82655	2023-01-01 01:17:00	Ariana Grande	Right There (feat. Big Sean)
82654	2023-01-01 01:17:00	Ariana Grande	Santa Baby
82653	2023-01-01 01:17:00	Kelly Clarkson	Santa, Can't You Hear Me
82650	2023-01-01 01:17:00	Ariana Grande	positions
82651	2023-01-01 01:17:00	Peach Pit	Being so Normal
...
59800	2023-12-07 21:13:00	Lana Del Rey	Ride
59802	2023-12-07 21:13:00	Childish Gambino	Les
59806	2023-12-07 21:14:00	Leonard Cohen	Thanks for the Dance
59805	2023-12-07 21:14:00	Ariana Grande	my hair
59807	2023-12-07 21:17:00	The Vaccines	Your Love Is My Favourite Band

	msPlayed
82655	22929.0
82654	12293.0
82653	325.0
82650	417.0
82651	2205.0
...	...
59800	1126.0
59802	2484.0
59806	9317.0
59805	23757.0
59807	14661.0

[156557 rows x 4 columns]

2.3 Deleting rows Often in Data Science, you will encounter when a row entry has the value *NaN*, indicating missing data. These entries can skew your analysis, leading to inaccurate conclusions. For this task, identify and remove any rows in your DataFrame that contain *NaN* values. Later in the course, you might encounter other techniques of dealing with missing data, typically

referred to as *data imputation*. Here, though, you are just supposed to delete the entire rows with missing data.

```
[247]: df_NaN = df_2023.isna()
```

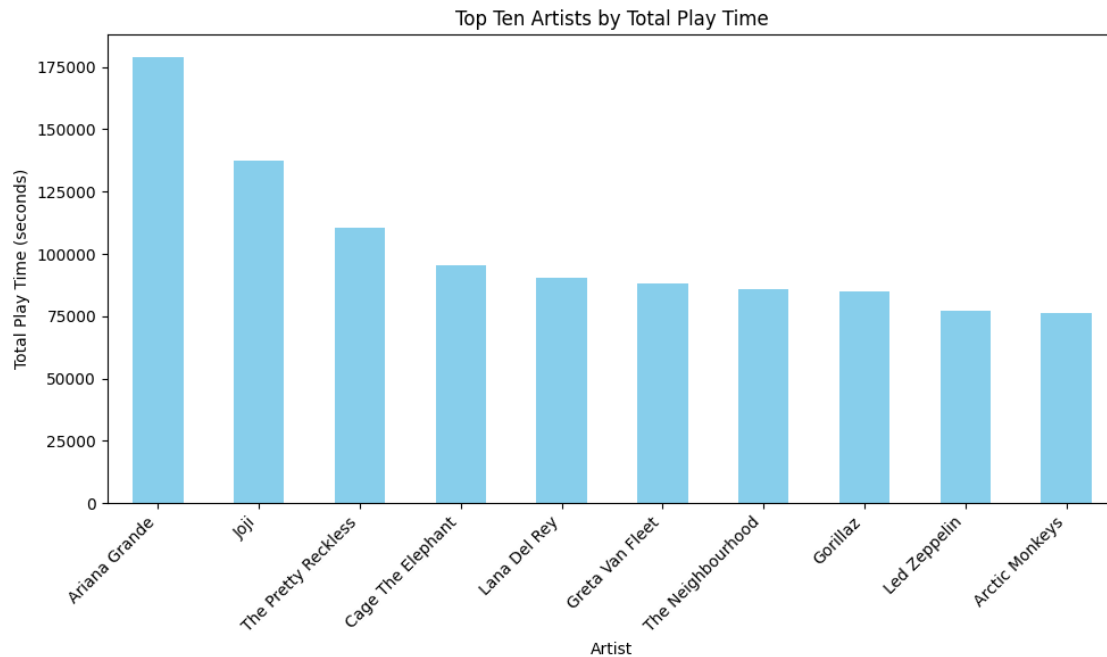
```
[248]: df_2023 = df_2023.dropna()
```

2.4 Convert from milliseconds to seconds From `msPlayed`, create a new column `secPlayed` with the data converted from milliseconds to seconds. Then delete the column `msPlayed`.

```
[ ]: df_2023['secPlayed'] = df_2023['msPlayed'] * 0.001  
df_2023 = df_2023.drop("msPlayed", axis='columns')
```

2.5 Finding top 10 favorite artists Find the top ten artists with the highest total play time (in seconds). Plot your findings in a bar graph.
(hint: start by creating a new DataFrame with only `artistName` and your time column. To proceed, you will also likely need the `groupby` command from Pandas.)

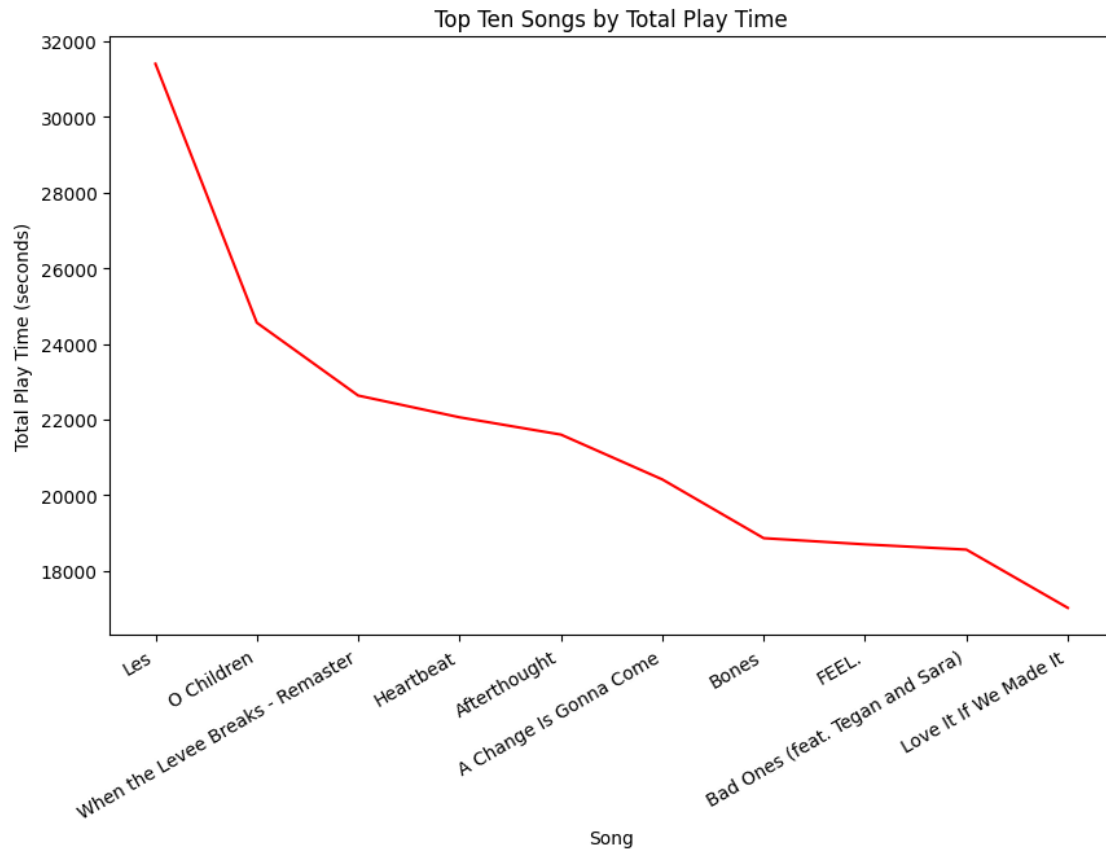
```
[250]: artist_playtime = df_2023.groupby('artistName')['secPlayed'].sum()  
top_ten_artists = artist_playtime.nlargest(10)  
  
plt.figure(figsize=(10, 6))  
top_ten_artists.plot(kind='bar', color='skyblue')  
plt.title('Top Ten Artists by Total Play Time')  
plt.xlabel('Artist')  
plt.ylabel('Total Play Time (seconds)')  
plt.xticks(rotation=45, ha='right')  
plt.tight_layout()  
plt.show()
```



2.6 Finding top 10 favorite songs Find the top ten songs with the highest play time. Create a graph visualizing the results.

```
[251]: song_playtime = df_2023.groupby('trackName')['secPlayed'].sum()
top_ten_songs = song_playtime.nlargest(10)

plt.figure(figsize=(10, 6))
top_ten_songs.plot(kind = 'line', color='red')
plt.title('Top Ten Songs by Total Play Time')
plt.xlabel('Song')
plt.ylabel('Total Play Time (seconds)')
plt.xticks(range(len(top_ten_songs)), top_ten_songs.index, rotation=30,
           ha='right')
plt.show()
```



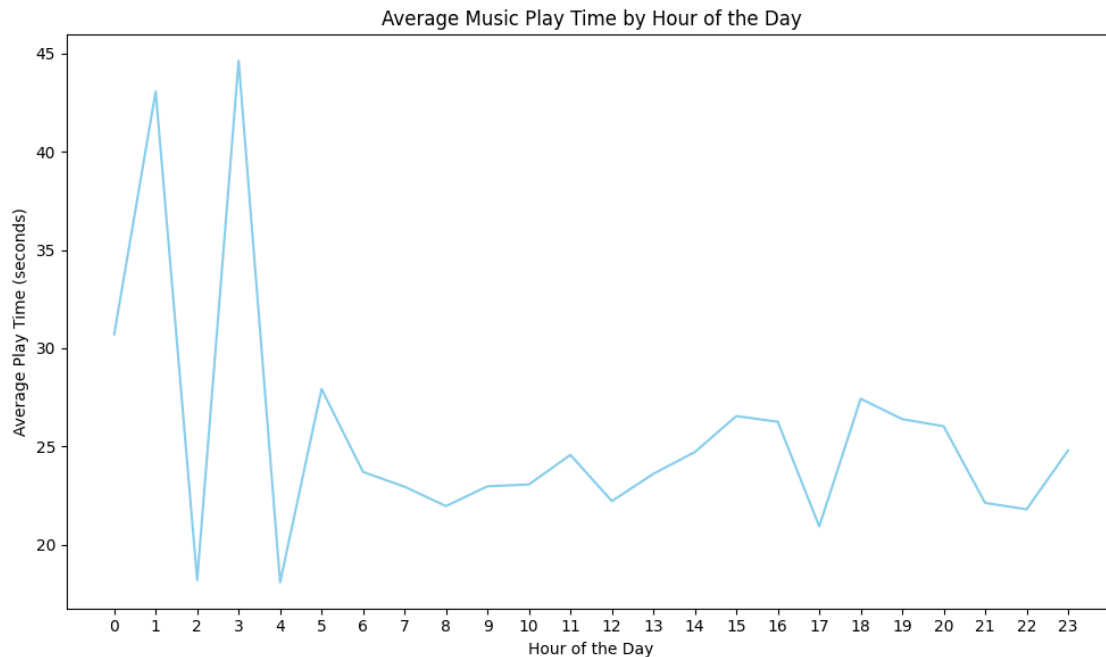
1.5 Part 3: Further analysis

3.0 Average listening time by hour Generate a plot that displays the average amount of time that music is played for each hour of the day.

```
[252]: df_2023['hour'] = df_2023['endTime'].dt.hour

avg_playtime_per_hour = df_2023.groupby('hour')['secPlayed'].mean()

plt.figure(figsize=(10, 6))
avg_playtime_per_hour.plot(kind='line', color='skyblue')
plt.title('Average Music Play Time by Hour of the Day')
plt.xlabel('Hour of the Day')
plt.ylabel('Average Play Time (seconds)')
plt.xticks(range(len(avg_playtime_per_hour)), avg_playtime_per_hour.index,
            rotation=0)
plt.tight_layout()
plt.show()
```

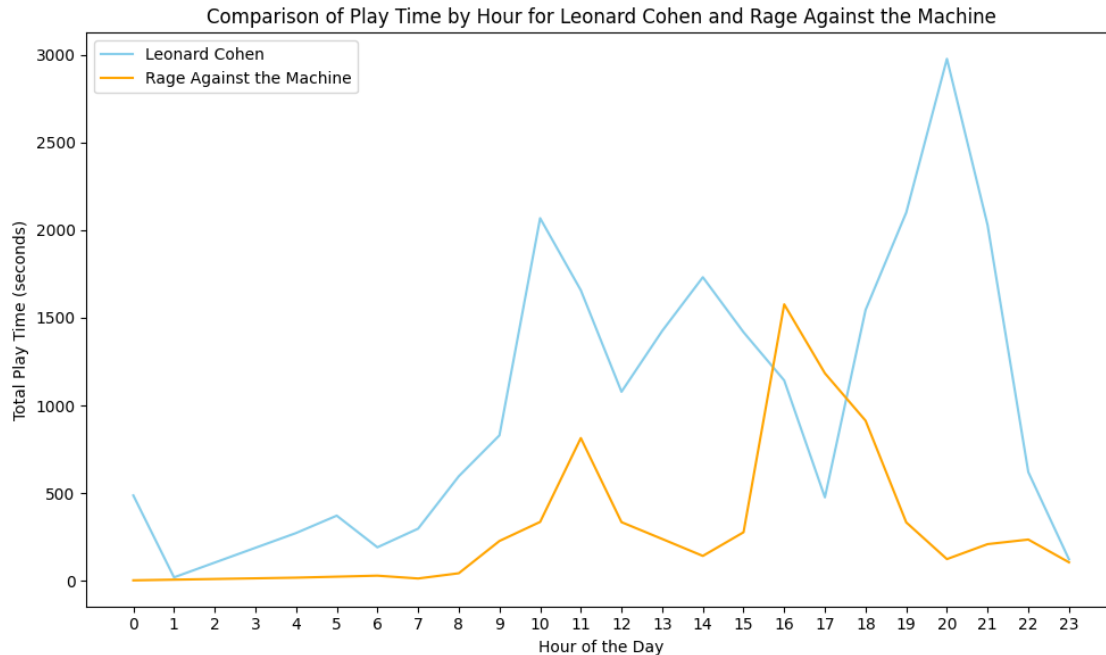


3.1 Morning music and evening music I think many people find that some types of music are more suitable for morning listening and some music is more suitable for evening listening. Create a plot that compares the play time of the artists *Leonard Cohen* and *Rage Against the Machine* on an hour-by-hour basis. See if there are any differences.

```
[253]: LC_df = df_2023[df_2023['artistName'] == 'Leonard Cohen']
RAtH_df = df_2023[df_2023['artistName'] == 'Rage Against The Machine']

cohen_playtime = LC_df.groupby('hour')['secPlayed'].sum()
RAtH_playtime = RAtH_df.groupby('hour')['secPlayed'].sum()

plt.figure(figsize=(10, 6))
plt.plot(cohen_playtime.index, cohen_playtime.values, color='skyblue',
        ↪label='Leonard Cohen')
plt.plot(RAtH_playtime.index, RAtH_playtime.values, color='orange', label='Rage
        ↪Against the Machine')
plt.title('Comparison of Play Time by Hour for Leonard Cohen and Rage Against
        ↪the Machine')
plt.xlabel('Hour of the Day')
plt.ylabel('Total Play Time (seconds)')
plt.xticks(df_2023['hour'].unique())
plt.legend()
plt.tight_layout()
plt.show()
```



3.2 Analysing skipped songs Determining whether a song was skipped or listened to can be challenging. For this analysis, we'll simplify by defining a skipped song as any track played for less than 30 seconds. Conversely, a song played for 30 seconds or more is considered listened to. Add a column to your DataFrame to reflect this criteria: set the value to 1 if the song was played for less than 30 seconds (indicating a skipped song), and 0 if it was played for 30 seconds or longer.

```
[254]: df_2023['skipped'] = (df_2023['secPlayed'] < 30).astype(int)
```

3.3 Plotting skipped songs Create a pie-chart that compares amount of skipped songs to amount of non-skipped songs.

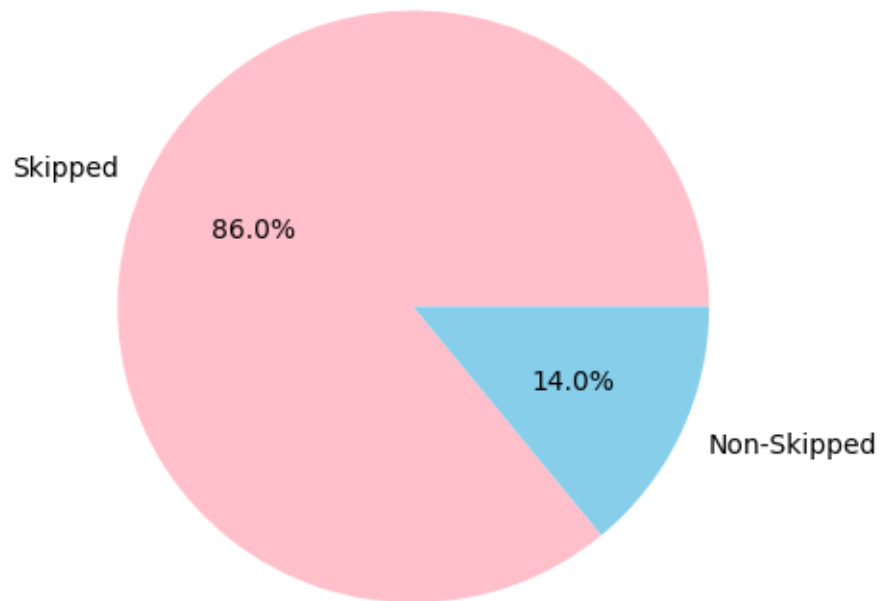
```
[255]: skipped_count = df_2023['skipped'].sum()
non_skipped_count = len(df_2023) - skipped_count

labels = ['Skipped', 'Non-Skipped']
sizes = [skipped_count, non_skipped_count]
colors = ['pink', 'skyblue']

plt.figure(figsize=(5, 5))
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%')
plt.title('Comparison of Skipped Songs vs Non-Skipped Songs')
```

```
[255]: Text(0.5, 1.0, 'Comparison of Skipped Songs vs Non-Skipped Songs')
```

Comparison of Skipped Songs vs Non-Skipped Songs



3.4 Artists by percentage of songs skipped For each artist in the dataset, calculate which percentage of their songs was skipped. Store this information in a new DataFrame called `df_skipped`. Store the percentage of skipped songs in a new column named `SkipRate`

Example: If an artist has **100** songs in your dataset and **25** of these were skipped, the percentage of skipped songs for this artist would be $\frac{25}{100} = 25\%$

```
[256]: artists_grouped = df_2023.groupby('artistName').agg({'skipped': 'sum',  
    ↳ 'secPlayed': 'count'})  
artists_grouped['SkipRate'] = (artists_grouped['skipped'] /  
    ↳ artists_grouped['secPlayed']) * 100  
df_skipped = artists_grouped[['SkipRate']]  
df_skipped
```

```
[256]:
```

	SkipRate
artistName	
10cc	67.857143
2Pac	86.159844
3 Doors Down	50.000000
4 Non Blondes	72.131148
50 Cent	67.857143

```
...
squeeda          66.666667
tenkousei.       100.000000
trxxshed         50.000000
xander.          37.500000
Édith Piaf       94.193548
```

```
[956 rows x 1 columns]
```

3.5 Comparing artists by skip-rate Find the **three** top artists with the lowest skip-rate and the **three** with the highest. Print their names, along with their skip-rate.

```
[257]: topp_3_high = df_skipped.nlargest(3, 'SkipRate')
topp_3_low = df_skipped.nsmallest(3, 'SkipRate')

print(topp_3_high)
print(topp_3_low)
```

```
                SkipRate
artistName
A Problem Squared    100.0
Acid Ghost           100.0
Albert Hammond Jr    100.0

                SkipRate
artistName
Gloria Gaynor        0.000000
Roc Boyz             11.111111
LACES                14.285714
```

1.6 Part 4: God Is a Data Scientist - The Ariana Deep-Dive

4.0 Ariana-DataFrame: Create a new DataFrame called *df_ariana*, containing only rows with music by Ariana Grande.

```
[258]: df_ariana = df_2023[df_2023['artistName'] == 'Ariana Grande']
```

4.1 Average skip rate Create a histogram of the distribution of the skip-rate values of the different artists in your DataFrame *df_skipped*, with skip rates on one axis and number of artists on the other.

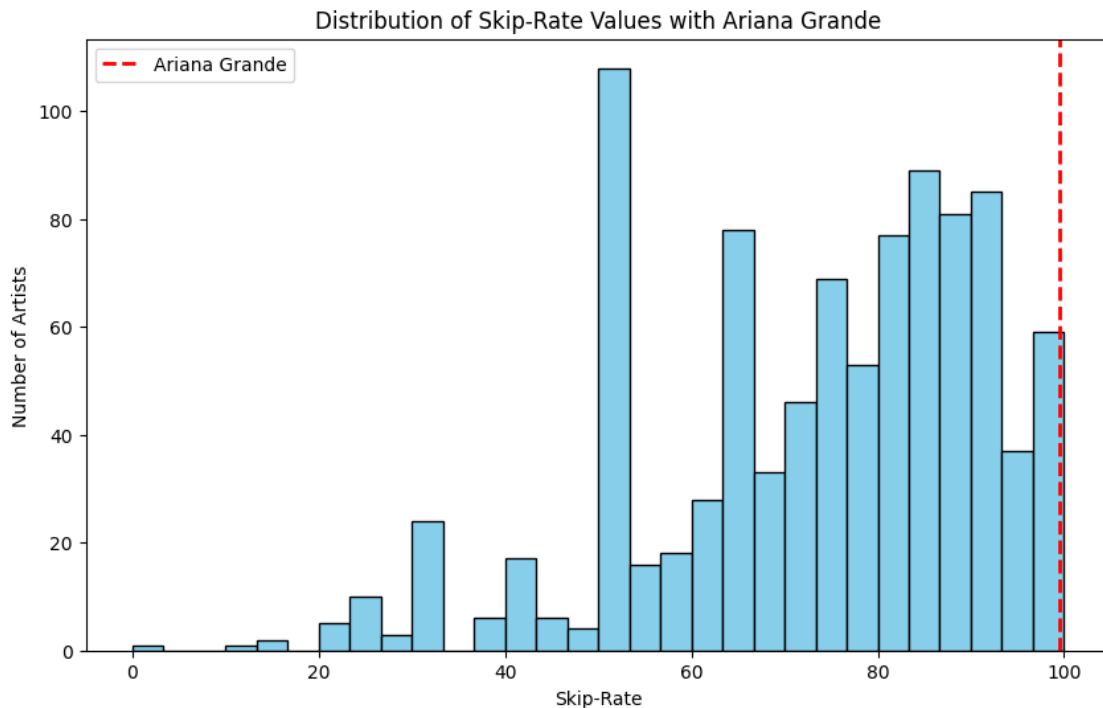
Then, retrieve the skip rate for Ariana Grande from your DataFrame *df_skipped*. Run the code in the cell below. Where on this distribution does Ariana Grande fall? Do I skip her songs more than average, or less?

```
[259]: plt.figure(figsize=(10, 6))
plt.hist(df_skipped['SkipRate'], bins=30, color='skyblue', edgecolor='black')

ariana_grande_skip_rate = df_skipped.loc['Ariana Grande', 'SkipRate']
```

```
plt.axvline(x=ariana_grande_skip_rate, color='red', linestyle='--', linewidth=2, label='Ariana Grande')
plt.title('Distribution of Skip-Rate Values with Ariana Grande')
plt.xlabel('Skip-Rate')
plt.ylabel('Number of Artists')
plt.legend()
plt.show()

print(f"Ariana Grande's skip rate: {ariana_grande_skip_rate}")
```



Ariana Grande's skip rate: 99.52939959662822

1.6.1 Part 4: Questions

Q1: Did I skip a lot of Ariana Grande's songs, or did I not, compared to the rest of the dataset? Yes, Ariana's skip rate was 99.5%, which is way more than average / Q2: What might be some possible reasons for Ariana Grande to be my nr.1 artist? Because, even though most of the songs were skipped, the total playtime by Artist in seconds where Ariana Grande as shown in task 2.5

[]: