Importing the downloaded json files

```python
import json
import pandas as pd
import glob

# Correct path with raw string and pattern
files = glob.glob(
    r"C:\Users\sachin\Downloads\my_spotify_data\Spotify Extended Streaming History\*.json"
)

data = []

for f in files:
    with open(f, "r", encoding="utf-8") as file:
        data.extend(json.load(file))

df = pd.DataFrame(data)
print(df.head(1))
print("Total records:", len(df))
```

Cleaning and basic formatting

Removing duplicates
Dropping unnecessary columns

```
df = df.drop_duplicates()
df.drop(labels='spotify_track_uri',axis =1,inplace =True)
df.drop([('incognito_mode'),('ip_addr')],axis =1,inplace =True)
df.drop(labels='offline_timestamp',axis =1,inplace =True)
df.drop(labels='conn_country',axis =1,inplace =True)
df.drop([('audiobook_title'),('audiobook_uri'),('audiobook_chapter_uri'),('audiobook_chapter_title') ],axis =1,inpla
df.drop(labels='spotify_episode_uri',axis =1,inplace =True)
```

Understanding data and again clean up

```
df['platform'].unique()
```

```
df.sample(5)
```

```
df.info()
```

Looking at unique platforms,sample of our dataframe.

```python
df.index.name ='index'

df.rename(columns={
    "master_metadata_track_name": "track_name",
    "master_metadata_album_artist_name": "artist",
    "master_metadata_album_album_name": "album"
}, inplace=True)
```

Renaming columns

```python
df['ts'] = pd.to_datetime(df['ts'], utc=True)
df['ts'] = df['ts'].dt.tz_convert('Asia/Kolkata')

df["date"] = df["ts"].dt.date
df["time"] = df["ts"].dt.time
df["year"] = df["ts"].dt.year
df["month"] = df["ts"].dt.month
df["day"] = df["ts"].dt.day
df["weekday"] = df["ts"].dt.weekday
df["hour"] = df["ts"].dt.hour
df["min_played"] = df["ms_played"] / (1000 * 60)
df["sec_played"] = df["ms_played"] / 1000
```

Creating new columns out of the present columns.

```python
df.drop([('ts'),('ms_played')],axis =1,inplace =True)
```

Again dropping some columns

Rearranging columns in our dataframe

```
df = df[['date','time',"min_played", "sec_played", "platform", "track_name", "artist", "album", "episode_name'
```

Mapping and replacing some values in the column for better and clean understanding

```
platform_map = {
    "Android OS 9 API 28 (realme, RMX1971)": "realme mobile",
    "Android OS 10 API 29 (realme, RMX1971)": "realme mobile",
    "Android OS 11 API 30 (realme, RMX1971)": "realme mobile",
    "Android OS 8.1.0 API 27 (LAVA, Z60s)": "lava mobile",
    "Android OS 7.0 API 24 (Xiaomi, Redmi Note 4)": "redmi mobile",
    'android':'realme mobile'
}

df["platform"] = df["platform"].replace(platform_map)
```

More understanding of our data

```
df['year'].value_counts()
```

```
import datetime
df_28 = df[df['date'] == datetime.date(2025, 10, 28)]
```

```
pd.set_option('display.max_rows', None)
```

```
df_2025 = df[df['year'] == 2025]
df_2025m = df_2025.groupby('month')['min_played'].sum().sort_values(ascending=False)
```

Early morning listening(2 am to 6am) usually when slept off or very rare party

```python
# Early morning = 2 AM to 6 AM
early = df[(df['hour'] >= 2) & (df['hour'] < 6)]
```

[102] ✓ 3.8s

```python
early_year = early.groupby('year')['min_played'].sum().reset_index()
print(early_year)
```

[103] ✓ 0.8s

```
   year  min_played
0  2020    5.502700
1  2021  962.388050
2  2022  850.074233
3  2023  637.730400
4  2024   11.572433
5  2025  122.372383
```

Years where early morning listening happened – maximum occurred in 2021, next 2022, very rarly happened in 2020 and 2024.

```python
early_top_days = (
    early.groupby('date')['min_played']
        .sum()
        .sort_values(ascending=False)
        .head(10)
)

print(early_top_days)
```

[105] ✓ 0.1s

```
date
2021-09-02    243.159500
2021-03-27    225.215650
2022-01-01    209.030517
2023-09-29    203.715000
2021-11-24    150.891850
2021-08-26    142.429200
2021-12-13    141.248550
2023-02-12    116.302183
2022-09-14    111.819617
2023-02-16    105.790817
Name: min_played, dtype: float64
```

Top days where early morning listening happened –

date	minutes
2021-09-02	243.159500
2021-03-27	225.215650
2022-01-01	209.030517
2023-09-29	203.715000
2021-11-24	150.891850
2021-08-26	142.429200
2021-12-13	141.248550
2023-02-12	116.302183
2022-09-14	111.819617
2023-02-16	105.790817

Maximum happened on 2nd September 2021

```python
early_weekday = (
    early.groupby('weekday')['min_played']
        .sum()
        .reset_index()
)

import calendar
early_weekday['weekday_name'] = early_weekday['weekday'].apply(lambda x: calendar.day_name[x])

print(early_weekday)
```

[106] 0.0s

	weekday	min_played	weekday_name
0	0	141.248550	Monday
1	1	314.061250	Tuesday
2	2	281.401750	Wednesday
3	3	562.474983	Thursday
4	4	392.346467	Friday
5	5	677.790467	Saturday
6	6	220.316733	Sunday

Most Early morning listening happened on Saturdays followed by Thursdays.

min_played	weekday_name
141.248550	Monday
314.061250	Tuesday
281.401750	Wednesday
562.474983	Thursday
392.346467	Friday
677.790467	Saturday
220.316733	Sunday

```python
len(early['date'].unique())
```

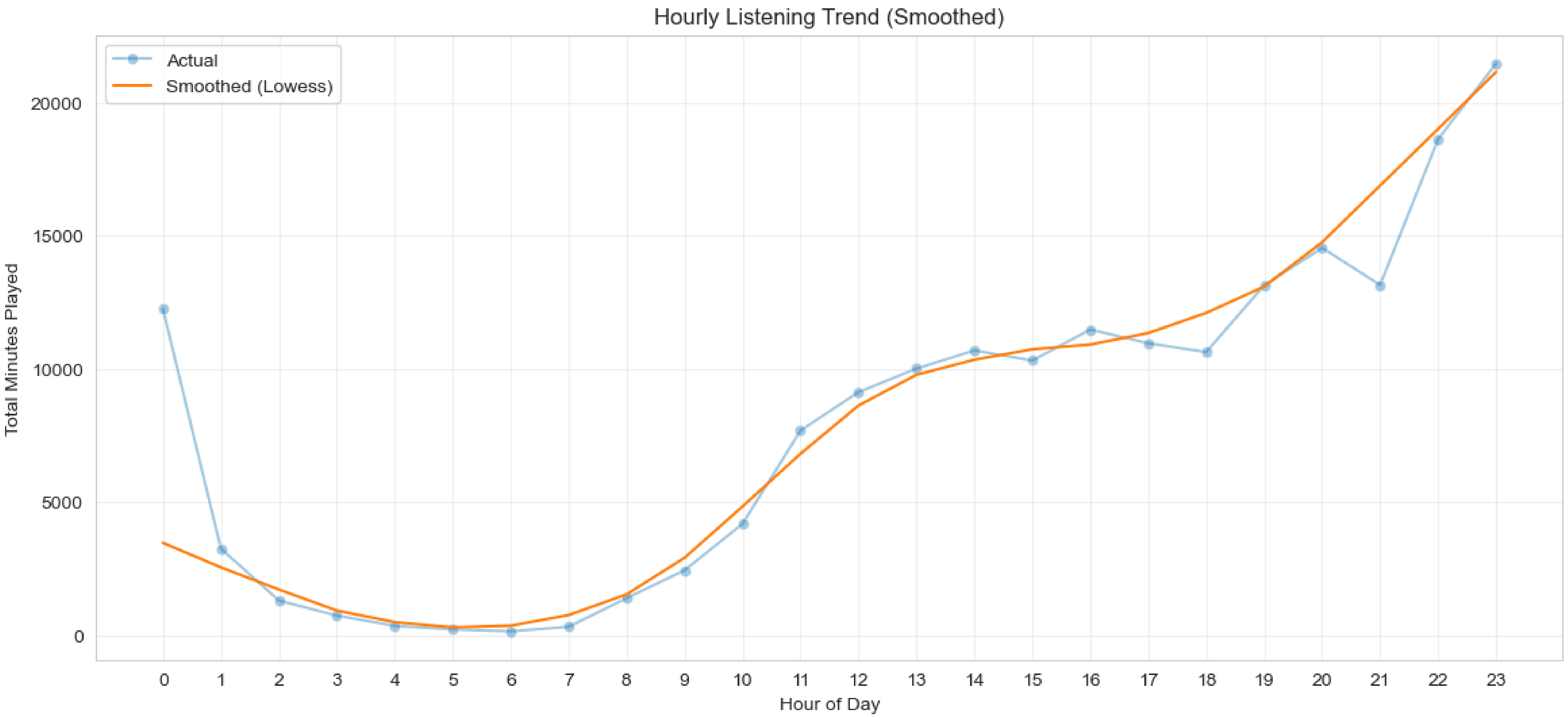
[108] 0.1s

41

Across all years, Early morning listening happened for 41 days

Hours trend

```python
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm

hourly = df.groupby('hour')['min_played'].sum().reset_index()

# LOWESS smoothing
smoothed = sm.nonparametric.lowess(hourly['min_played'], hourly['hour'], frac=0.3)

plt.figure(figsize=(14,6))
sns.lineplot(x=hourly['hour'], y=hourly['min_played'], alpha=0.4, label="Actual", marker='o')
sns.lineplot(x=smoothed[:,0], y=smoothed[:,1], label="Smoothed (Lowess)", linewidth=1.5)

plt.title("Hourly Listening Trend (Smoothed)")
plt.xlabel("Hour of Day")
plt.ylabel("Total Minutes Played")
plt.xticks(range(24))
plt.grid(alpha=0.3)
plt.legend()
plt.show()
```


Hourly Listening Trend (Smoothed)
Actual
Smoothed (Lowess)
Total Minutes Played
0
5000
10000
15000
20000
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
Hour of Day

12 AM (00:00) — HUGE spike
•This is your **second-highest listening hour of the entire day**.

8 AM – 12 PM: Gradual ramp-up
•From 8 AM your listening **increases steadily**.

12 PM – 6 PM: Afternoon plateau
•Listening stays **consistently high**.

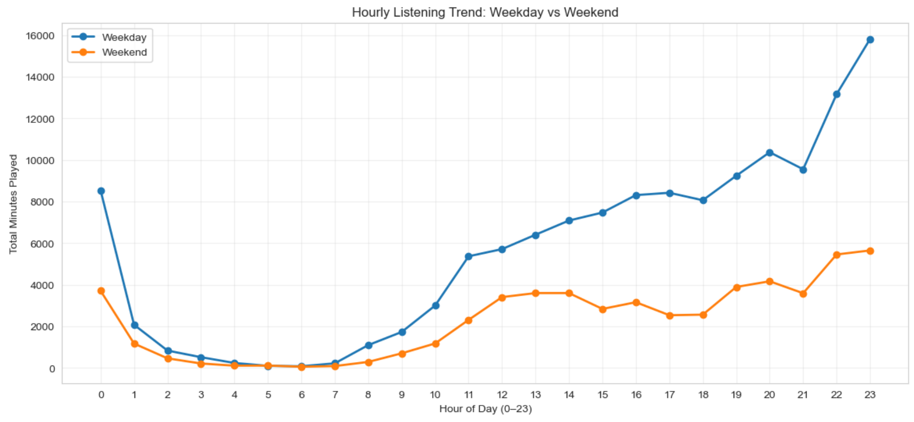
6 PM – 10 PM: Strong evening rise
•From 6 PM onwards, listening rises again

10 PM – 12 AM: Maximum listening of the day
•10 PM climbs sharply
•11 PM is very high

11 PM — Peak listening of the entire day

Weekdays vs Weekends listening pattern

```python
df['weekend'] = df['weekday'].apply(lambda x: 1 if x >= 5 else 0)
hour_weekend = df.groupby(['hour', 'weekend'])['min_played'].sum().reset_index()
weekday_data = hour_weekend[hour_weekend['weekend'] == 0]
weekend_data = hour_weekend[hour_weekend['weekend'] == 1]
import matplotlib.pyplot as plt

plt.figure(figsize=(14,6))

plt.plot(weekday_data['hour'], weekday_data['min_played'],
         marker='o', label='Weekday', linewidth=2)

plt.plot(weekend_data['hour'], weekend_data['min_played'],
         marker='o', label='Weekend', linewidth=2)

plt.title("Hourly Listening Trend: Weekday vs Weekend")
plt.xlabel("Hour of Day (0–23)")
plt.ylabel("Total Minutes Played")
plt.xticks(range(24))
plt.legend()
plt.grid(alpha=0.3)
plt.show()
```

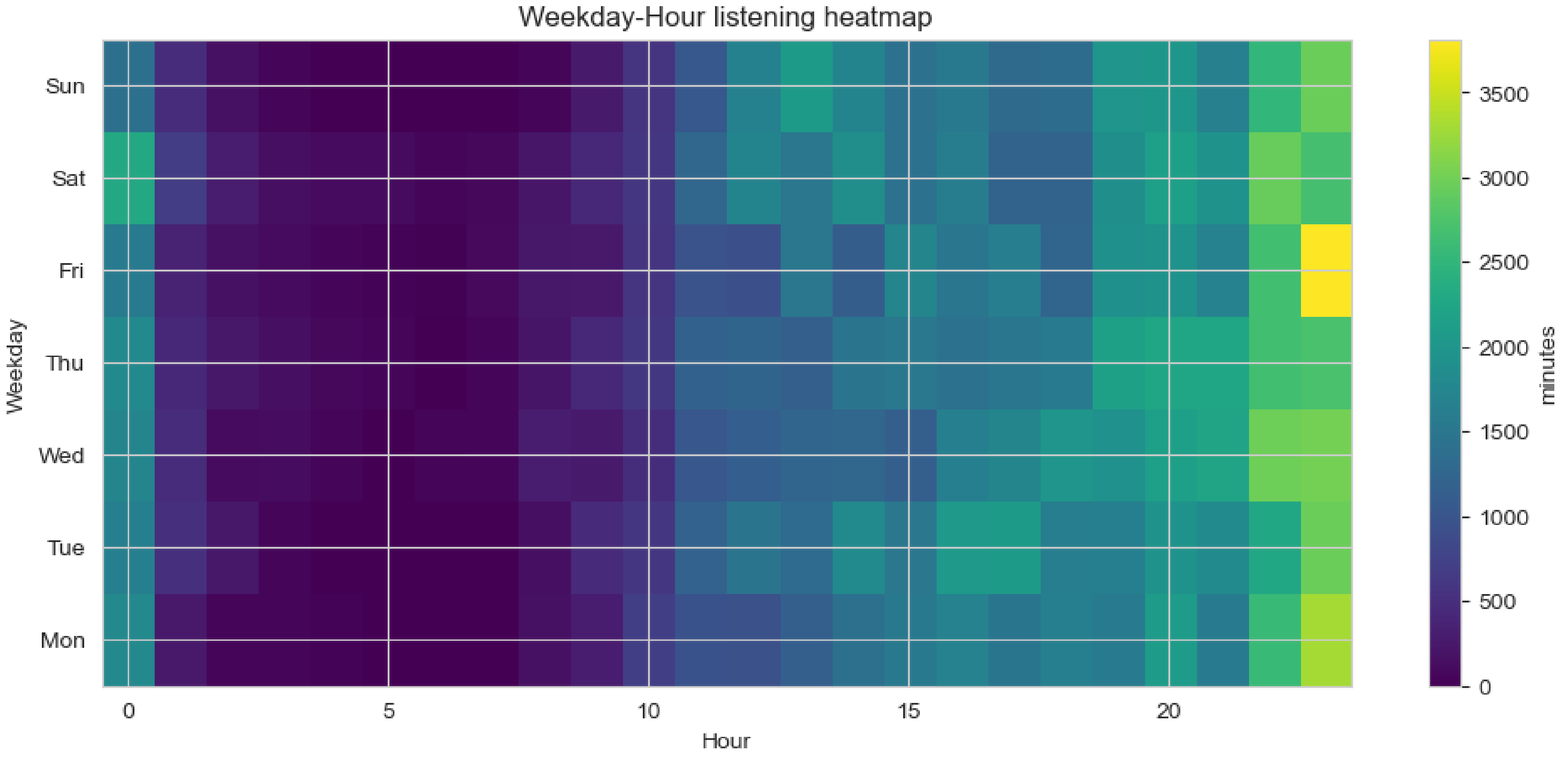


Observation –
on weekends,there is a dip in listening during evenings.Both during weekdays and weekends, the listening is very less in the morning hours, grows more in afternoons,and peak at night hours.

Weekday and hours relation

```
pivot = df.groupby(['weekday','hour'])['min_played'].sum().unstack(fill_value=0)
plt.figure(figsize=(12,5))
plt.imshow(pivot, aspect='auto', origin='lower',cmap ='viridis')
plt.colorbar(label='minutes')
plt.yticks(range(7), ['Mon','Tue','Wed','Thu','Fri','Sat','Sun'])
plt.xlabel('Hour'); plt.ylabel('Weekday'); plt.title('Weekday-Hour listening heatmap')
plt.show()
```

✓ 1.8s



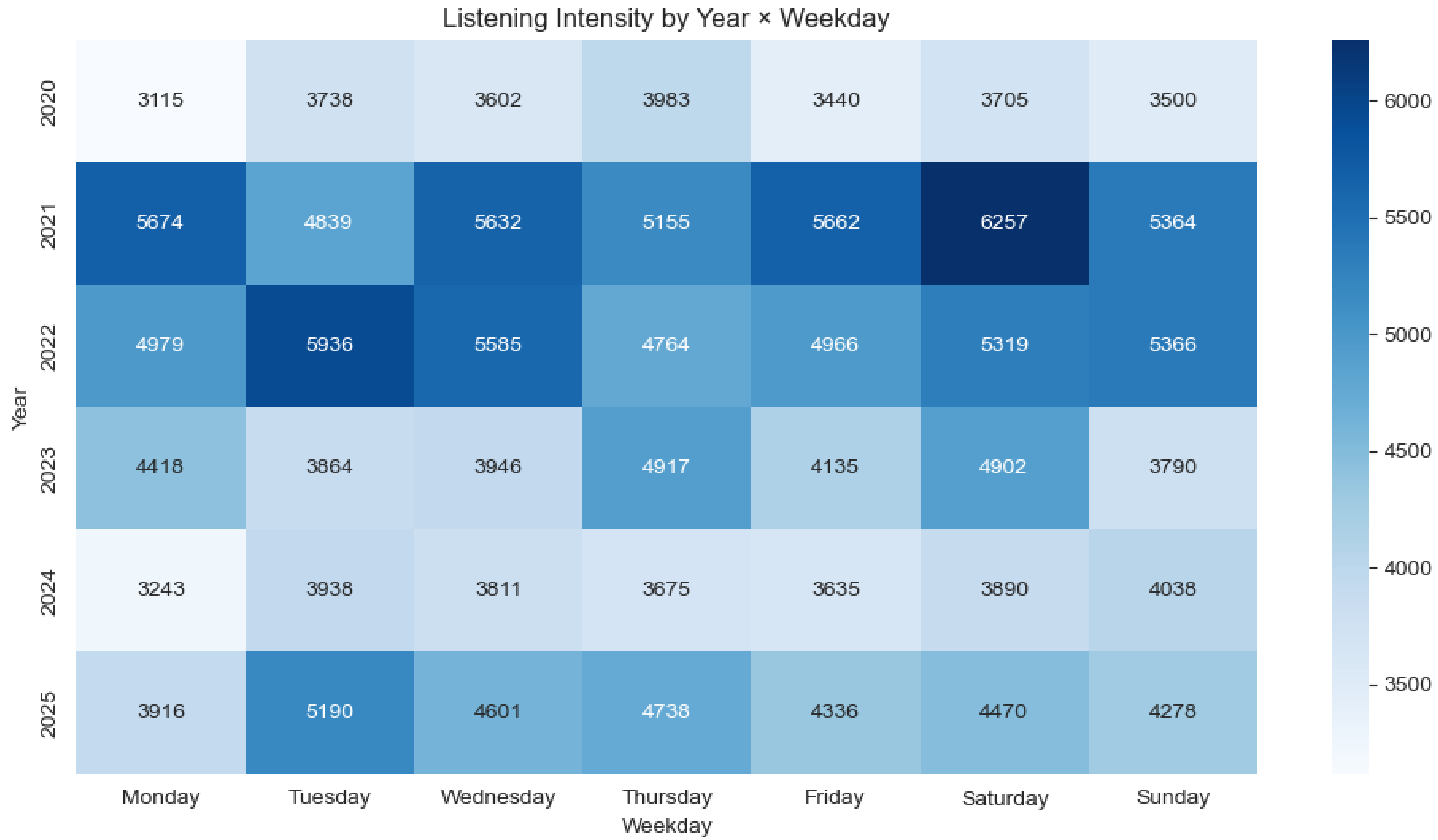
1. Nights Are Your Peak Time (20:00–23:00)

•Every single weekday has a strong band of **dark yellow/green** around **8 PM–11 PM**.

Friday shows the **single highest hour overall** (23rd hour).

Early Morning Listening Is Almost Zero

•Hours **3 AM to 7 AM** are nearly blank on all days.



Listening Intensity by Year × Weekday
Year
2020
2021
2022
2023
2024
2025
Weekday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
3115
3738
3602
3983
3440
3705
3500
5674
4839
5632
5155
5662
6257
5364
4979
5936
5585
4764
4966
5319
5366
4418
3864
3946
4917
4135
4902
3790
3243
3938
3811
3675
3635
3890
4038
3916
5190
4601
4738
4336
4470
4278
6000
5500
5000
4500
4000
3500

```python
df.groupby('weekday')['min_played'].sum() \
  .reset_index(name='total_minutes') \
  .sort_values(by='total_minutes', ascending=False)
```

[116] ✓ 0.1s

	weekday	total_minutes
5	5	28542.778133
1	1	27503.847667
3	3	27231.407817
2	2	27177.904300
6	6	26335.605417
4	4	26174.553733
0	0	25345.466550

Most listening of songs happened on Saturdays then on Tuesdays, Least on Mondays.

weekday	Total minutes
Sunday	26335
Monday	25345
Tuesday	27503
Wednesday	27177
Thursday	27231
Friday	26174
Saturday	28542

Listening intensity year and weekdays relation

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import calendar

# Convert date to datetime
df['date'] = pd.to_datetime(df['date'])

# Map weekday numbers to names
df['weekday_name'] = df['weekday'].apply(lambda x: calendar.day_name[x])

# Group by year & weekday
pivot = (
    df.groupby(['year', 'weekday_name'])['min_played']
      .sum()
      .reset_index()
      .pivot(index='year', columns='weekday_name', values='min_played')
)

# Reorder columns to Monday → Sunday
order = ['Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday']
pivot = pivot[order]

# Plot heatmap
plt.figure(figsize=(12,6))
sns.heatmap(pivot, annot=True, fmt='.0f', cmap='Blues')
plt.title("Listening Intensity by Year × Weekday")
plt.xlabel("Weekday")
plt.ylabel("Year")
plt.show()
```

2021 is your peak listening year

Saturday (6257 min) → your *most intense listening day* across all years

Streak where songs where heard for min 4 hrs a day(240mins) – [streak with a threshold]

```python
THRESHOLD = 240  # change as you like

daily = df.groupby('date')['min_played'].sum().sort_index()

intense = (daily > THRESHOLD)

# Detect streaks of True values
streaks = []
start = None
length = 0
prev_date = None

for date, val in intense.items():
    if val:
        if start is None:
            start = date
            length = 1
        elif (date - prev_date).days == 1:
            length += 1
        else:
            streaks.append((start, prev_date, length))
            start = date
            length = 1
    else:
        if start is not None:
            streaks.append((start, prev_date, length))
            start = None
            length = 0
    prev_date = date

if start is not None:
    streaks.append((start, prev_date, length))

intense_streaks = pd.DataFrame(streaks, columns=['start', 'end', 'length']).sort_values('length', ascending=False)
print(intense_streaks.head(10))
```

start	end	length
2025-09-21	2025-09-26	6
2025-10-28	2025-10-31	4
2021-03-24	2021-03-27	4
2025-08-05	2025-08-07	3
2025-10-22	2025-10-24	3
2020-10-31	2020-11-01	2
2023-03-19	2023-03-20	2
2025-08-18	2025-08-19	2
2021-02-27	2021-02-28	2
2025-07-23	2025-07-24	2

Longest streak happened in 📅 21 Sept 2025 → 26 Sept 2025 (6 days)

```python
df['date'] = pd.to_datetime(df['date'])
df['year'] = df['date'].dt.year
df['dayofyear'] = df['date'].dt.dayofyear

day_sum = df.groupby(['year','dayofyear'])['min_played'].sum().reset_index()

plt.figure(figsize=(14,6))
pivot = day_sum.pivot(index='year', columns='dayofyear', values='min_played')

sns.heatmap(pivot, cmap='Blues')
plt.title("Listening Intensity per Day of Year")
plt.xlabel("Day of Year")
plt.ylabel("Year")
plt.show()
```

Listening intensity per day of year.

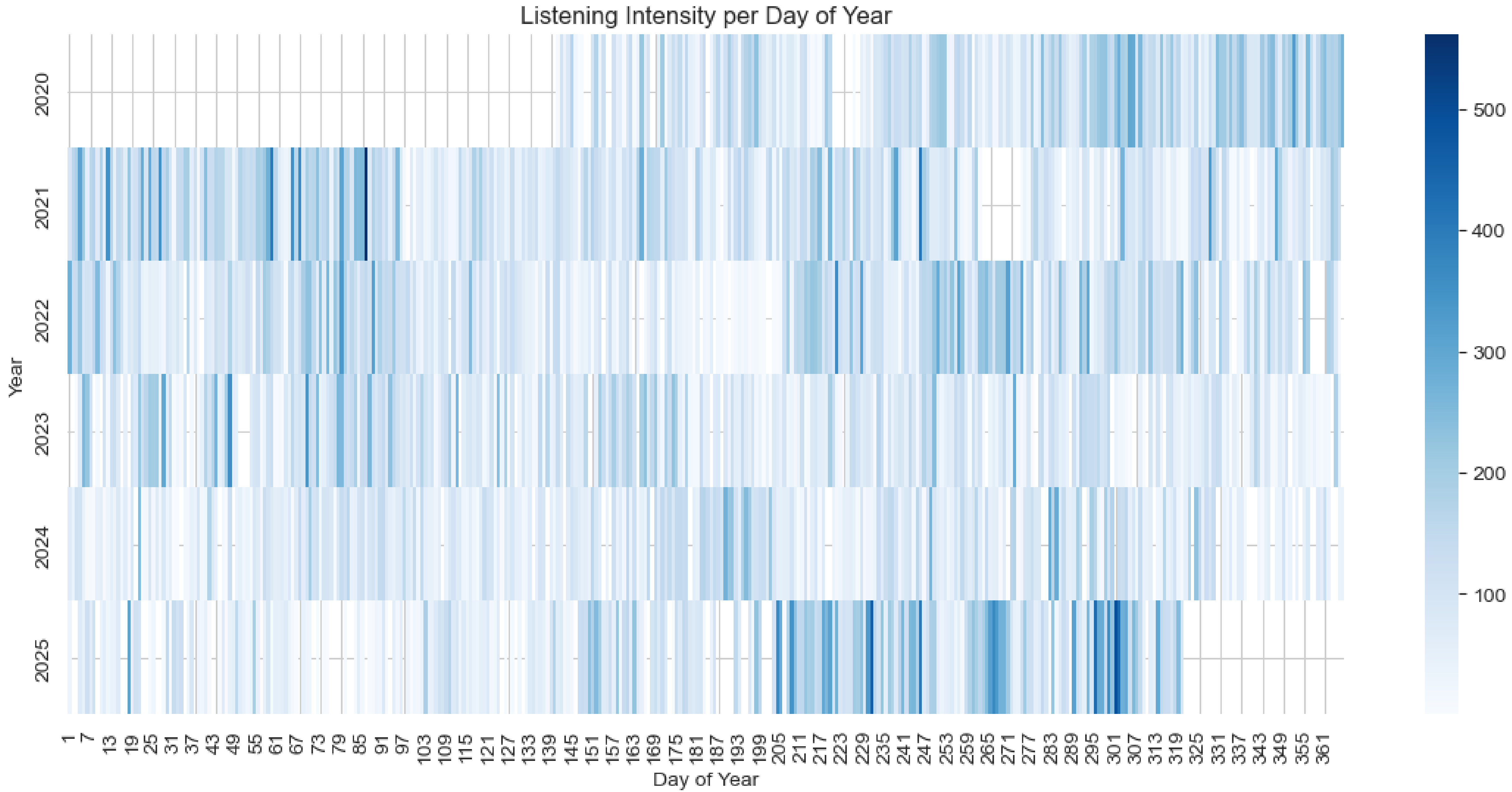
```python
top_days_minutes = (
    df.groupby('date')['min_played']
      .sum()
      .sort_values(ascending=False)
      .head(20)
)

print(top_days_minutes)
```

[81] ✓ 0.3s

Top 5 days of listening -

```
Date
2021-03-27  560.712467
2025-10-28  493.459733
2025-08-19  459.451083
2025-10-29  432.483283
2025-10-22  428.606867
```

Highest listening time was on 27nd March 2021


Listening Intensity per Day of Year
Year
2020
2021
2022
2023
2024
2025
Day of Year
1
7
13
19
25
31
37
43
49
55
61
67
73
79
85
91
97
103
109
115
121
127
133
139
145
151
157
163
169
175
181
187
193
199
205
211
217
223
229
235
241
247
253
259
265
271
277
283
289
295
301
307
313
319
325
331
337
343
349
355
361
500
400
300
200
100

Average minute per day

```python
# Step 1: total minutes per day
daily_totals = (
    df.groupby(['year', 'date'])['min_played']
      .sum()
      .reset_index()
)

# Step 2: average daily listening per year
avg_daily_per_year = (
    daily_totals.groupby('year')['min_played']
    .mean()
    .reset_index(name='avg_minutes_per_day')
)

print(avg_daily_per_year)
```

[130] ✓ 11.1s

```
...     year  avg_minutes_per_day
    0   2020           118.317679
    1   2021           110.870204
    2   2022           107.307253
    3   2023            89.736315
    4   2024            78.767930
    5   2025           114.654689
```

⭐ **Average minutes played on a day *when you actually listened***

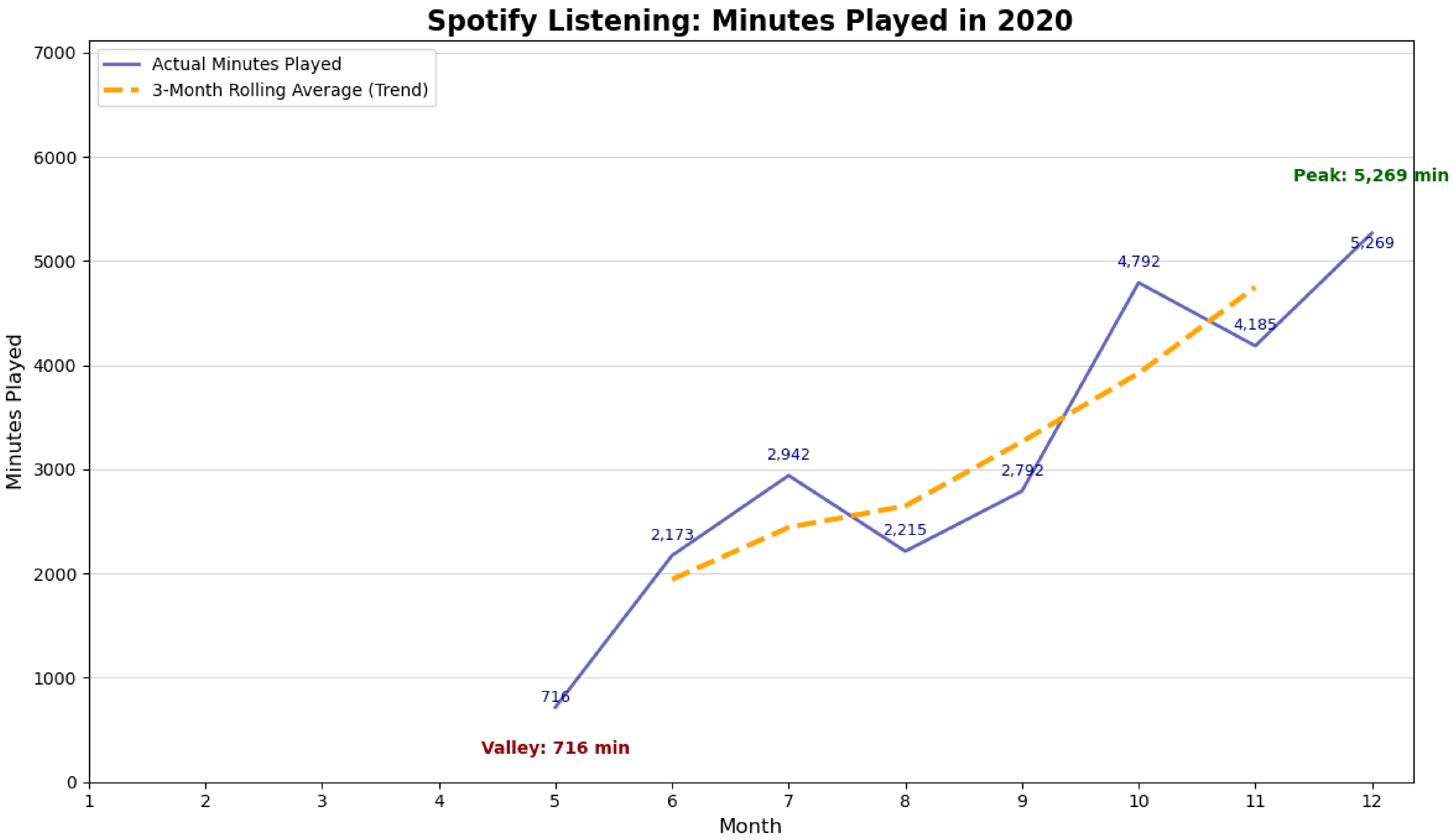
year	Avg time per day
2020	118
2021	111
2022	107
2023	90
2024	79
2025	115

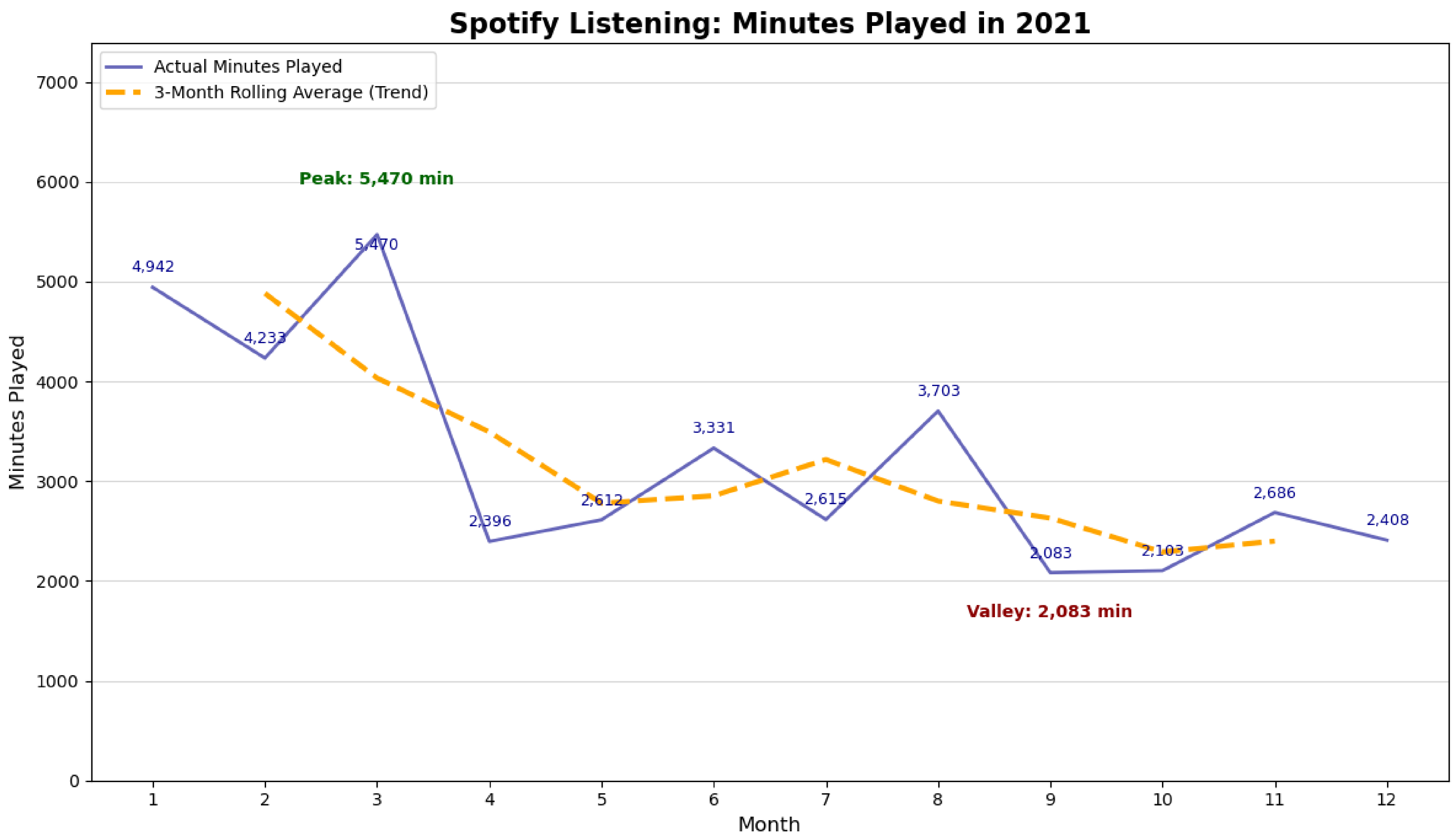
Creating year wise time spent in music graphs

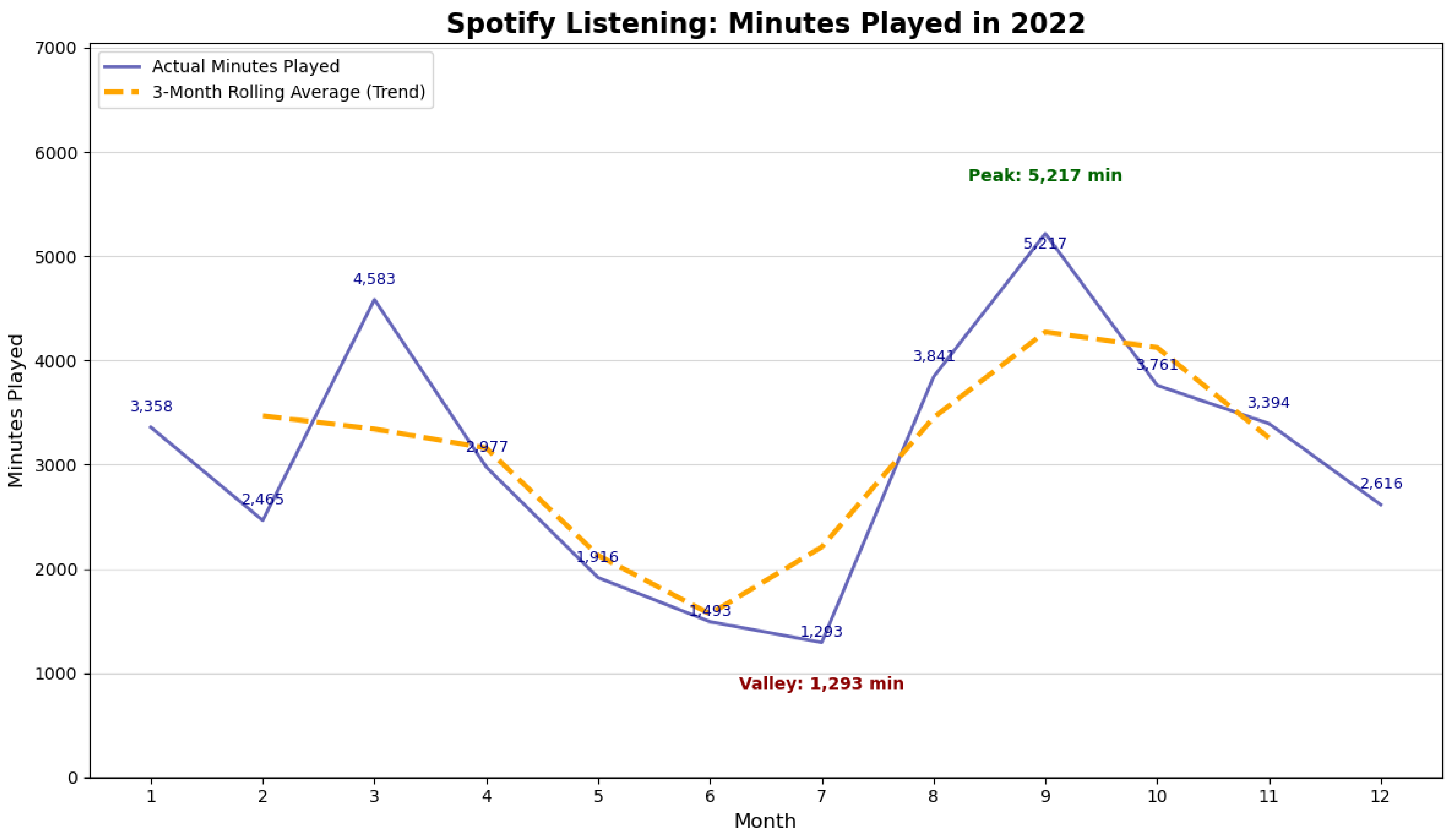
```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

def generate_monthly_listening_plot(df, year):
    """
    Filters df for a given year, aggregates minutes by month,
    calculates rolling average, and generates an insightful Seaborn line plot.
    """

    # 1. Filter Data for the Specific Year
    df_year = df[df['year'] == year].copy()

    if df_year.empty:
        print(f"No data found for the year {year}. Skipping plot.")
        return

    # 2. Aggregate Minutes by Month (Chronological Time Series)
    # Grouping by 'month' and summing 'min_played'
    df_monthly = (
        df_year.groupby('month')['min_played']
        .sum()
        .sort_index()  # Crucial for chronological plotting
        .rename('Minutes Played')
        .to_frame()
    )
    # 3. Add Insight: 3-Month Rolling Average
    df_monthly['3-Month Rolling Avg'] = df_monthly['Minutes Played'].rolling(window=3, center=True).mean()

    # --- 4. Seaborn Plotting ---
    plt.figure(figsize=(12, 7))

    # Plot Raw Data (Volatility)
    sns.lineplot(
        data=df_monthly,
        x=df_monthly.index,
        y='Minutes Played',
        label='Actual Minutes Played',
        linewidth=2,
        color='darkblue',
        alpha=0.6
    )
    # Plot Rolling Average (Trend)
    sns.lineplot(
        data=df_monthly,
        x=df_monthly.index,
        y='3-Month Rolling Avg',
        label='3-Month Rolling Average (Trend)',
        linewidth=3,
        linestyle='--',
        color='orange'
    )

    # --- 5. Add Data Labels and Annotations ---
    peak_month = df_monthly['Minutes Played'].idxmax()
    peak_value = df_monthly['Minutes Played'].max()

    valley_month = df_monthly['Minutes Played'].idxmin()
    valley_value = df_monthly['Minutes Played'].min()

    # Data Labels with Smart Offset
    for month, value in df_monthly['Minutes Played'].items():
        y_offset = 150 # Default offset: up

        # Adjust offset if value is very low (near the bottom) or at the peak for better visual separation
        if value < df_monthly['Minutes Played'].mean() * 0.5:
            y_offset = 50 # Smaller positive offset for low values
        elif month == peak_month:
            y_offset = -150 # Move peak label down to avoid overlapping with annotation

        plt.text(month, value + y_offset,
                 f'{value:,.0f}', color='darkblue', ha='center', fontsize=9, fontweight='light')

    # Peak Annotation (Lifted high)
    plt.text(peak_month, peak_value + 500,
             f'Peak: {peak_value:,.0f} min', color='darkgreen', fontweight='bold', ha='center', fontsize=10)

    # Valley Annotation (Moved low)
    plt.text(valley_month, valley_value - 450,
             f'Valley: {valley_value:,.0f} min', color='darkred', fontweight='bold', ha='center', fontsize=10)
```

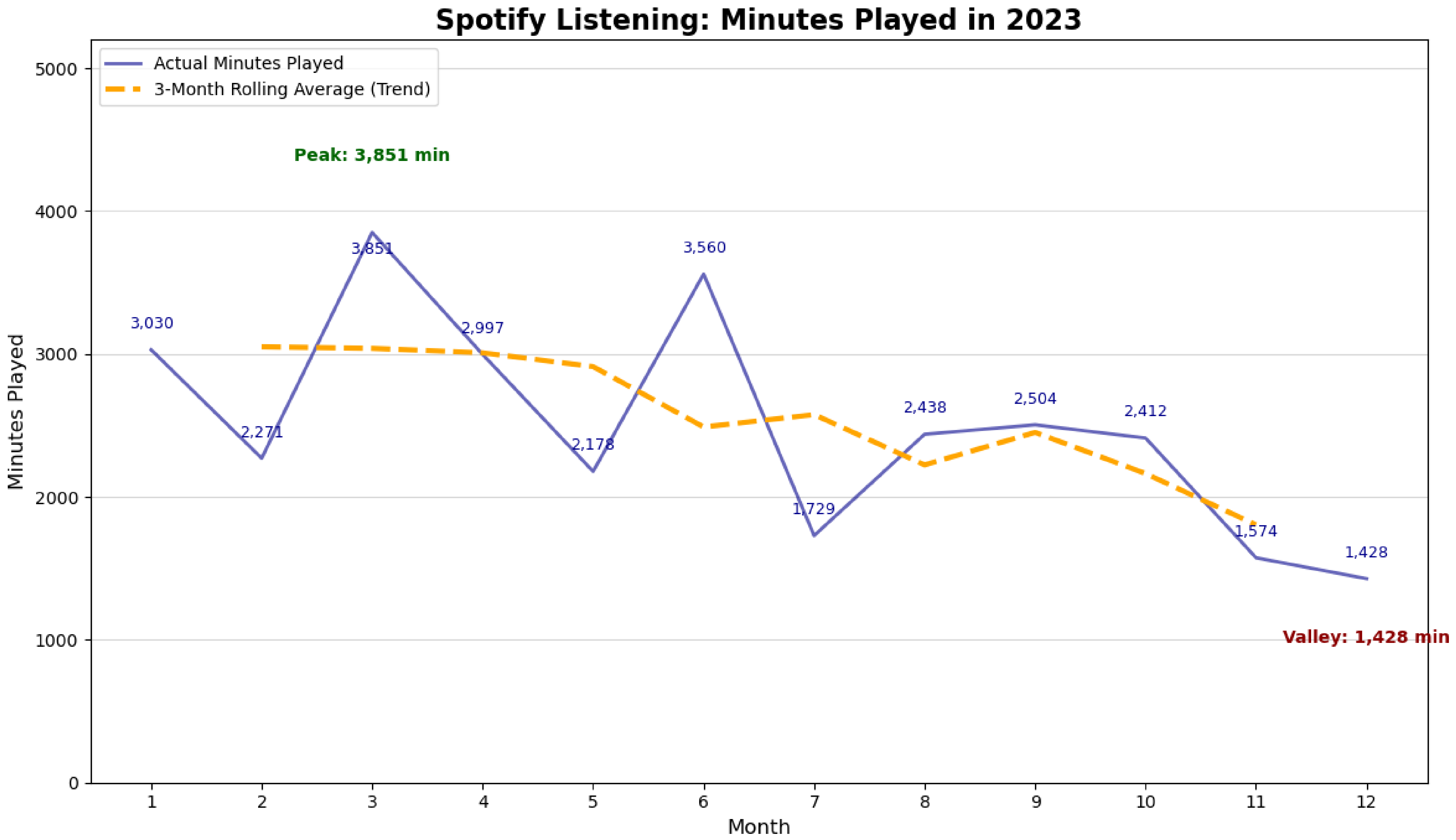
```python
    # --- 6. Customization and Save ---
    plt.title(f'Spotify Listening: Minutes Played in {year}', fontsize=16, fontweight='bold')
    plt.xlabel('Month', fontsize=12)
    plt.ylabel('Minutes Played', fontsize=12)

    # Ensure all month numbers (1-12) are shown on the x-axis
    plt.xticks(range(1, 13))

    # Set Y-limit dynamically based on the max value of the current year
    plt.ylim(0, df_monthly['Minutes Played'].max() * 1.35)

    plt.grid(axis='y', linestyle='-', alpha=0.5)
    plt.legend(loc='upper left', fontsize=10)
    plt.tight_layout()
    plt.savefig(f'spotify_listening_{year}.png')
    plt.close()

# ======================================================================
# EXECUTION LOOP
# ======================================================================

# List of the five unique years you provided
years_to_plot = [2020, 2021, 2022, 2023,2024, 2025]

# Run the plotting function for all five years
for year in years_to_plot:

    generate_monthly_listening_plot(df, year)

print(f"Finished generating {len(years_to_plot)} plots. Check your working directory for the image files: spotify_listening_2020.png through spotify_
```

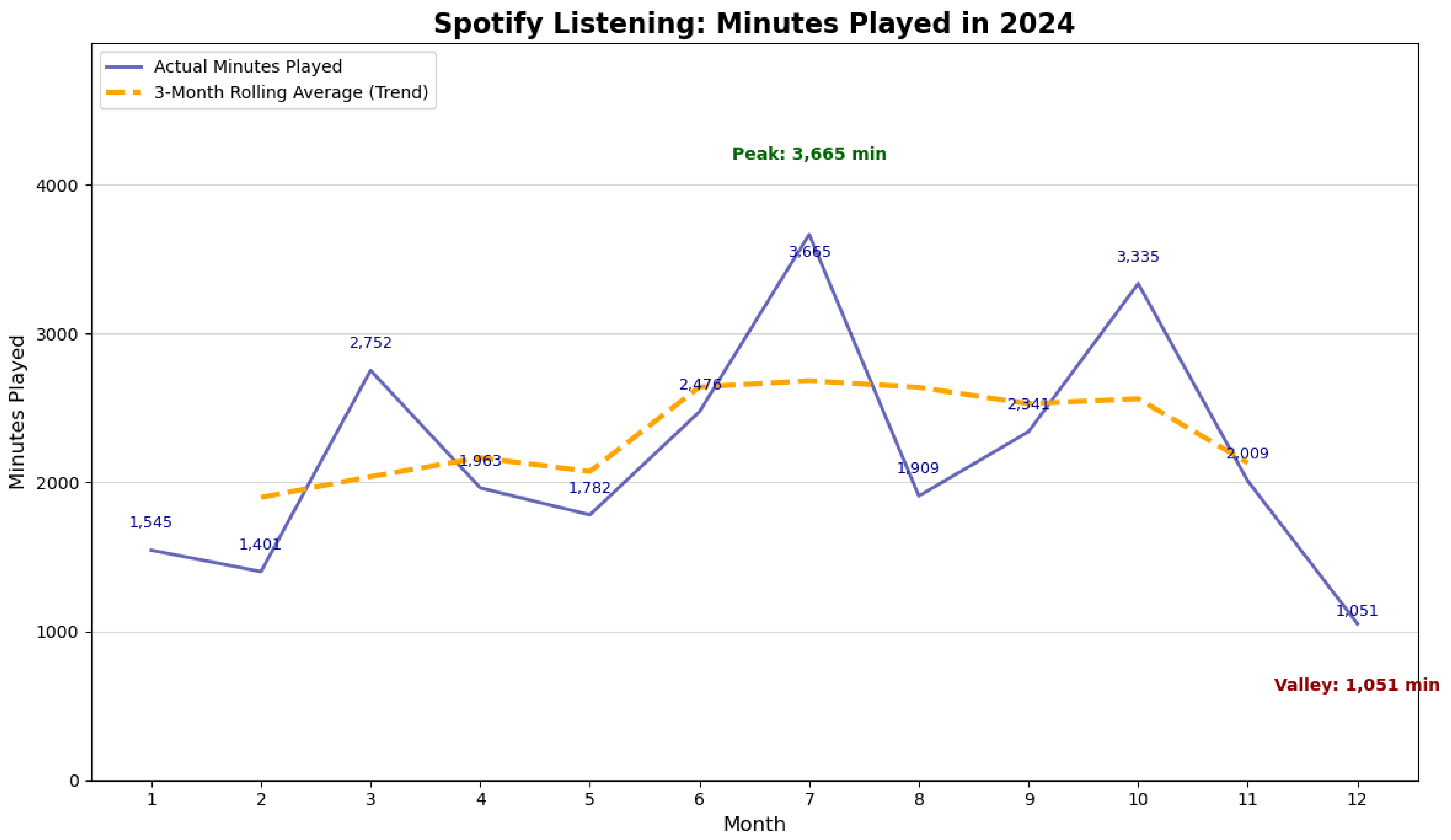
YEAR WISE LISTENING TIME

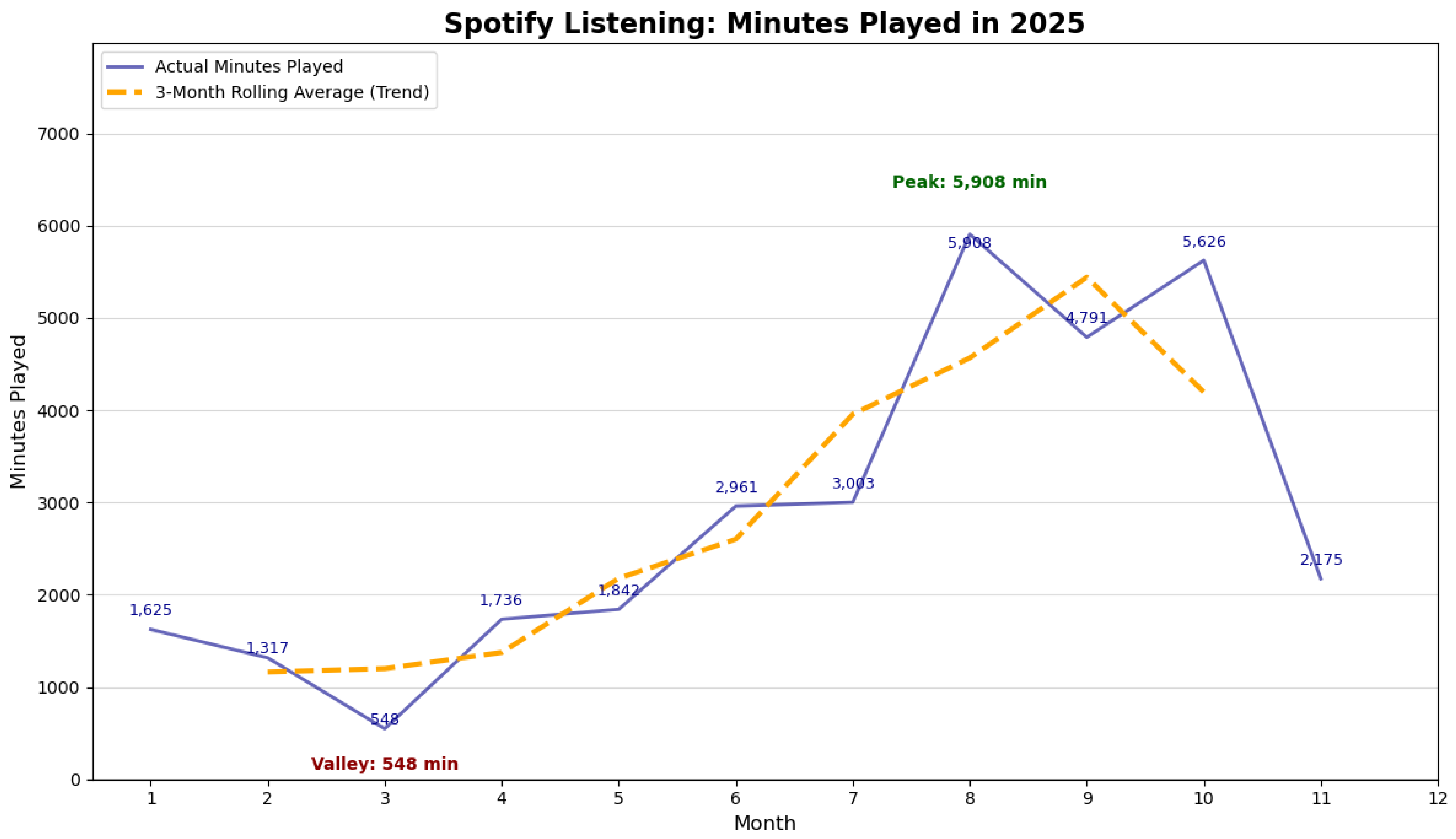



Spotify Listening: Minutes Played in 2021
Actual Minutes Played
3-Month Rolling Average (Trend)
Peak: 5,470 min
Valley: 2,083 min
4,942
4,233
5,470
2,396
2,612
3,331
2,615
3,703
2,083
2,103
2,686
2,408
Minutes Played
Month
0
1000
2000
3000
4000
5000
6000
7000
1
2
3
4
5
6
7
8
9
10
11
12


Spotify Listening: Minutes Played in 2022
Actual Minutes Played
3-Month Rolling Average (Trend)
Peak: 5,217 min
Valley: 1,293 min
3,358
2,465
4,583
2,977
1,916
1,493
1,293
3,841
5,217
3,761
3,394
2,616
Minutes Played
Month
0
1000
2000
3000
4000
5000
6000
7000
1
2
3
4
5
6
7
8
9
10
11
12


Spotify Listening: Minutes Played in 2023
Actual Minutes Played
3-Month Rolling Average (Trend)
Peak: 3,851 min
Valley: 1,428 min
Minutes Played
Month
0
1000
2000
3000
4000
5000
1
2
3
4
5
6
7
8
9
10
11
12
3,030
2,271
3,851
2,997
2,178
3,560
1,729
2,438
2,504
2,412
1,574
1,428


Spotify Listening: Minutes Played in 2024
Actual Minutes Played
3-Month Rolling Average (Trend)
Peak: 3,665 min
Valley: 1,051 min
1,545
1,401
2,752
1,963
1,782
2,476
3,665
1,909
2,341
3,335
2,009
1,051
Minutes Played
Month
0
1000
2000
3000
4000
1
2
3
4
5
6
7
8
9
10
11
12



Spotify Listening: Minutes Played in 2025
Actual Minutes Played
3-Month Rolling Average (Trend)
Peak: 5,908 min
Valley: 548 min
1,625
1,317
548
1,736
1,842
2,961
3,003
5,908
4,791
5,626
2,175
Minutes Played
Month

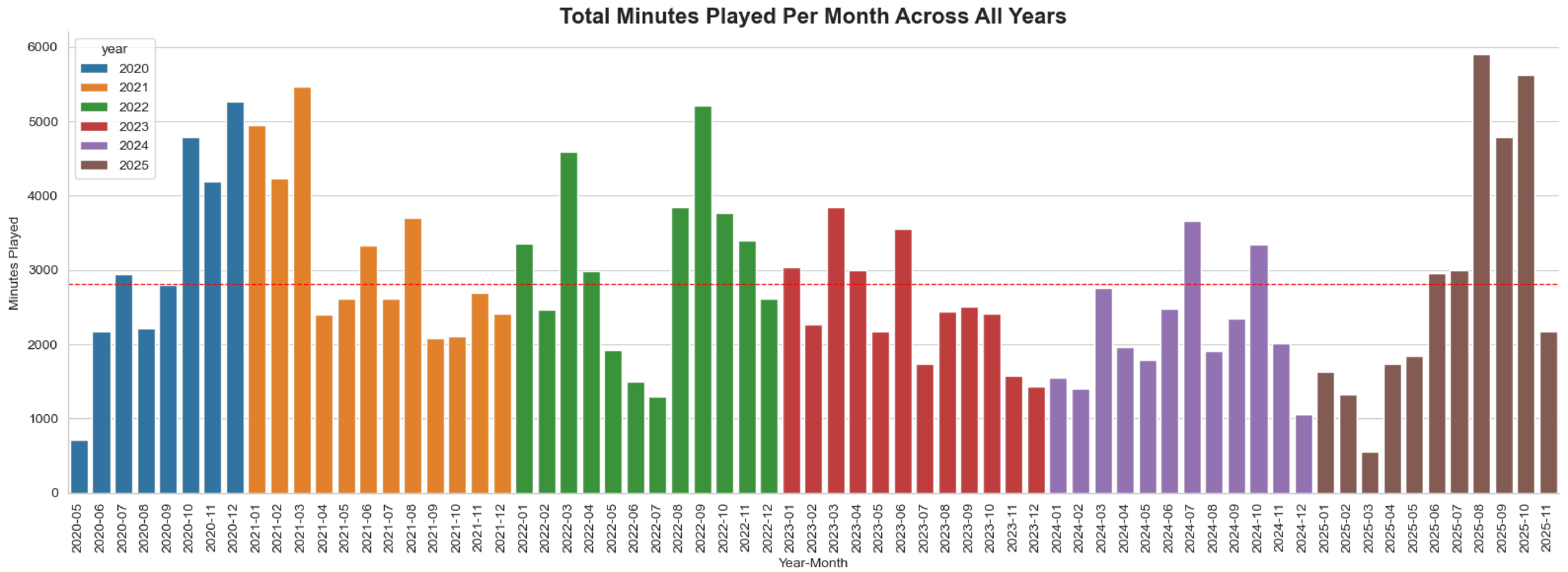
All years - line

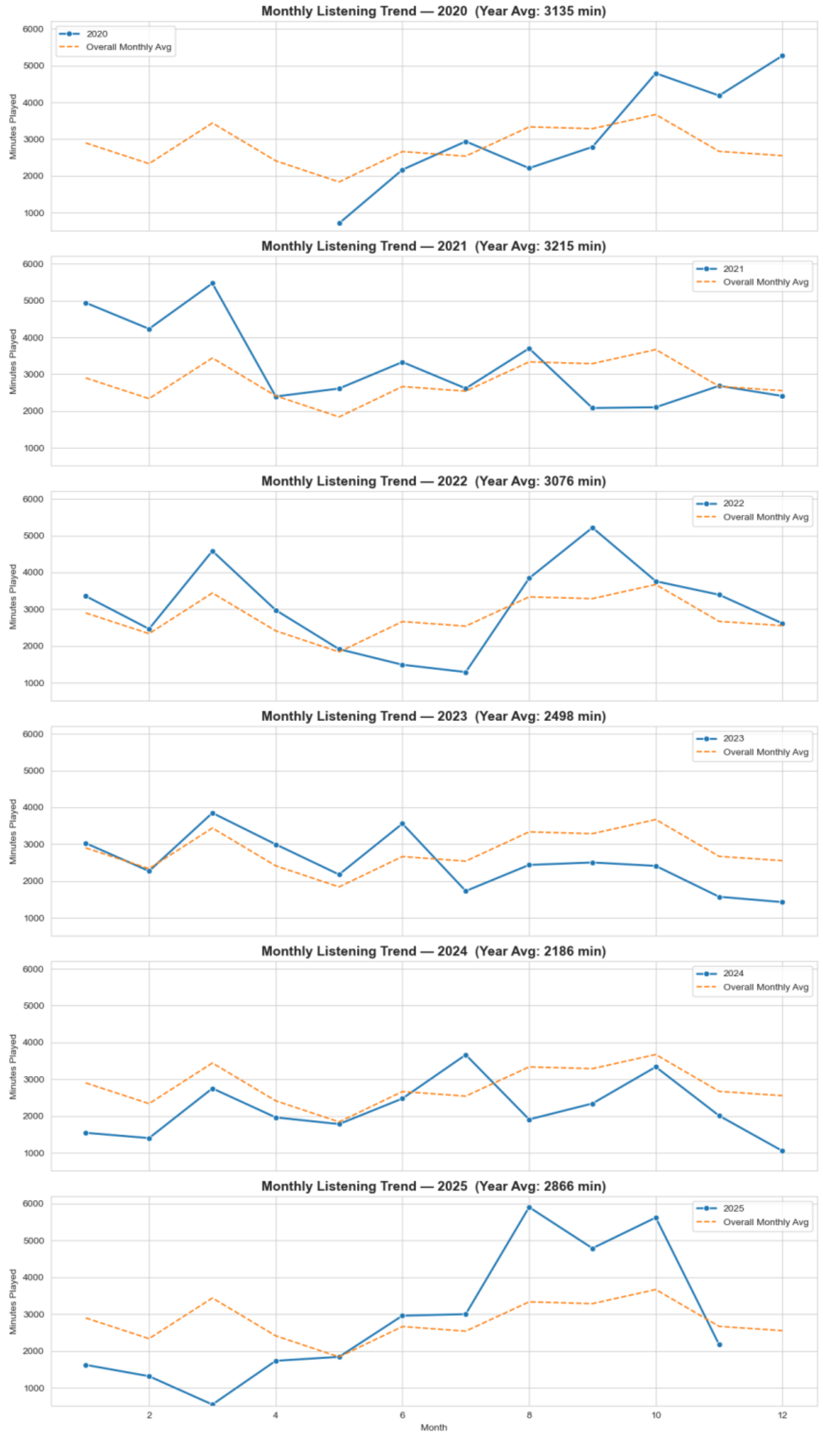
```python
monthly = (
    df.groupby(['year','month'])['min_played']
      .sum()
      .reset_index()
)

# Compute yearly averages
yearly_avg = (
    monthly.groupby('year')['min_played']
           .mean()
           .reset_index()
           .rename(columns={'min_played': 'year_avg'})
)

# Overall monthly average
overall_avg = (
    monthly.groupby('month')['min_played']
           .mean()
           .reset_index()
           .rename(columns={'min_played': 'overall_avg'})
)

# Merge yearly avg + overall avg into monthly data
monthly = monthly.merge(yearly_avg, on='year').merge(overall_avg, on='month')

# Compute global y-axis range
ymin = monthly['min_played'].min() * 0.95
ymax = monthly['min_played'].max() * 1.05

# Plotting
sns.set_style("whitegrid")
unique_years = sorted(monthly['year'].unique())

fig, axes = plt.subplots(len(unique_years), 1, figsize=(12, 3.5 * len(unique_years)), sharex=True)
```

```python
for i, year in enumerate(unique_years):
    ax = axes[i] if len(unique_years) > 1 else axes

    temp = monthly[monthly['year'] == year]

    # Yearly line
    sns.lineplot(
        data=temp,
        x='month',
        y='min_played',
        marker='o',
        linewidth=2,
        ax=ax,
        label=f'{year}'
    )

    # Overall avg line
    sns.lineplot(
        data=overall_avg,
        x='month',
        y='overall_avg',
        linestyle='--',
        ax=ax,
        label='Overall Monthly Avg'
    )

    # Title with per-year average
    ax.set_title(f"Monthly Listening Trend – {year}  (Year Avg: {temp['year_avg'].iloc[0]:.0f} min)",
                 fontsize=14, weight='bold')

    ax.set_ylabel("Minutes Played")
    ax.set_ylim(ymin, ymax)
    ax.legend()

plt.xlabel("Month")
plt.tight_layout()
plt.show()
```

All years –bar

```python
monthly = (
    df.groupby(['year','month'])['min_played']
      .sum()
      .reset_index()
)

# Create a combined label for x-axis
monthly['year_month'] = monthly['year'].astype(str) + "-" + monthly['month'].astype(str).str.zfill(2)

# Compute overall average monthly value
overall_avg = monthly['min_played'].mean()

plt.figure(figsize=(16, 6))

sns.barplot(
    data=monthly,
    x='year_month',
    y='min_played',
    hue='year',          #  Color by year
    dodge=False,         #  Bars stay in original order
    palette='tab10'
)
#  Overall average horizontal line
plt.axhline(overall_avg, linestyle='--', linewidth=0.9, label=f"Overall Avg: {overall_avg:.0f}", color='red')


plt.title("Total Minutes Played Per Month Across All Years", fontsize=16, weight='bold')
plt.xlabel("Year-Month")
plt.ylabel("Minutes Played")

plt.xticks(rotation=90)
sns.despine()
plt.tight_layout()
plt.show()
```


Total Minutes Played Per Month Across All Years
year
2020
2021
2022
2023
2024
2025
Minutes Played
Year-Month
0
1000
2000
3000
4000
5000
6000
2020-05
2020-06
2020-07
2020-08
2020-09
2020-10
2020-11
2020-12
2021-01
2021-02
2021-03
2021-04
2021-05
2021-06
2021-07
2021-08
2021-09
2021-10
2021-11
2021-12
2022-01
2022-02
2022-03
2022-04
2022-05
2022-06
2022-07
2022-08
2022-09
2022-10
2022-11
2022-12
2023-01
2023-02
2023-03
2023-04
2023-05
2023-06
2023-07
2023-08
2023-09
2023-10
2023-11
2023-12
2024-01
2024-02
2024-03
2024-04
2024-05
2024-06
2024-07
2024-08
2024-09
2024-10
2024-11
2024-12
2025-01
2025-02
2025-03
2025-04
2025-05
2025-06
2025-07
2025-08
2025-09
2025-10
2025-11



ALL YEARS –(LINE AND BAR)

1. Long-Term Volume Shift (Total Minutes Played Per Year)

The series of line graphs (Image 1) and the bar chart (Image 3) both confirm a fundamental change in your overall listening volume:

•**Decline in Average Listening (Image 1):** Your average monthly listening time was highest during **2020 (3,135 min)** and **2021 (3,215 min)**. It then dropped sharply in the subsequent years:

- **2023:** 2,498 min
- **2024:** 2,186 min (The lowest average year)
- **2025:** 2,866 min (A rebound year, but below the 2020/2021 peak).

2. Monthly Volatility and Extreme Seasonality

The monthly data reveals an increasing reliance on specific, intense listening periods:

•**Increased Volatility (Image 1 & 3):** Your listening trend is becoming much more **erratic** year-over-year. The bars/lines in 2020 and 2021 generally stay near their average, but in **2024 and 2025**, you see huge spikes and deep valleys.

Seasonal trend -

The Mid-Year Spike (Months 7-10): Across all graphs, there is a recurring and intensifying spike between **Month 7 (July)** and **Month 10 (October)**.

•**2025 Peak (Image 2 & 3):** This is the most dramatic observation. Your peak listening occurred in **August 2025 (Month 8)**, reaching nearly **6,000 minutes**.

•**Deep Valleys:** Your lowest listening months are consistently in the **first quarter (Months 1-4)**, with the **lowest point ever** being **Month 3, 2025** (around 500 minutes).

•**Inference:** You are shifting from being a stable, consistent listener to a **highly seasonal listener**.

The 2025 Rebound

•**Average Recovery (Image 1):** After hitting the lowest yearly average in 2024 (2,186 min), the **2025 average has jumped back up to 2,866 min**.

(based on count)Top 10 songs(all time)

```python
top20_songs_count = (
    df['track_name']
      .value_counts()
      .head(10)
      .reset_index(name='play_count')
      .rename(columns={'index':'track_name'})
)

print(top20_songs_count)
```

[135] ✓ 12.5s

```
                                 track_name  play_count
0              Pookkalae Sattru Oyivedungal         183
1  Darkhaast (feat. Arijit Singh, Sunidhi Chauhan)  147
2                                 Heartless         141
3                               Do It Again         139
4                                 Enna Sona         129
```

```
track_name                    play_count
0 Pookkalae Sattru Oyivedungal 183
1 Darkhaast                    147
2 Heartless                    141
3 Do It Again                  139
4 Enna Sona                    129
5 Hosanna                      128
6 Jannatein Kahan              128
7 Munbe Vaa                    125
8 Five More Hours              123
9 Heart Attack                 118
```

```python
top5_songs_each_year = (
    df.groupby(['year', 'track_name'])
        .size()
        .reset_index(name='play_count')
        .sort_values(['year', 'play_count'], ascending=[True, False])
        .groupby('year')
        .head(5)
)

top5_songs_each_year
```

✓ 1.9s

	year	track_name	play_count
1546	2020	Tera Yaar Hoon Main (From "Sonu Ke Titu Ki Swe...	67
157	2020	Bandook Meri Laila	66
1374	2020	Savage Love (Laxed - Siren Beat)	61
567	2020	Hairaani	60

2020

song	No of plays
Tera yaar hoon main	67
Bandook meri laila	66
Savage love	61
hairaani	60
Zara zara	59

2022

song	No. of plays
Pookkalae satru oyivedungal	88
Go crazy	53
enadhuyire	52
Falak tak	49
saware	44

2021

song	No. of plays
hosanna	79
Munbe vaa	58
bujji	50
Kangal irandaal	46
Jashn e bahaara	45

2023

song	No. of plays
Vilambara idaiveli	75
Enna sona	70
Jimmiki ponnu	58
Aedho saigirai	55
Do it again	48

2024

song	No. of plays
heartless	98
fein	70
Angel numbers	67
Heart attack	57
Tum todo na	56

2025

song	No. of plays
Fire & desire	56
Hey daddy	55
Un-thinkable	51
Into you	43
finesse	39

Hindi songs dominated 2020,in 2021 we can see tamil songs domination

2022 – again tamil and hindi songs domination

2023 – tamil songs dominate

2024 – English songs dominate

2025 – English songs dominate

```
top10_artists = (
    df.groupby('artist')['track_name']
      .count()
      .sort_values(ascending=False)
      .head(10)
      .reset_index(name='play_count')
)

print(top10_artists)
```

Comparing all the years, AR Rahman has been in top 5 except 2025.Pritam,Chris Brown and Drake are also present in top list.

(based on count)Top 10 artist overall

-

	Artist	play_count
1	A.R. Rahman	3182
2	Pritam	2696
3	Arijit Singh	1995
4	Chris Brown	1807
5	Anirudh Ravichander	1536
6	Yuvan Shankar Raja	1334
7	Atif Aslam	1038
8	The Weeknd	830
9	Vishal-Shekhar	695
10	Jason Derulo	647

Finding top 5 artists of each year and their percent of total plays.

```python
top5 = (
    df.groupby(['year','artist'])['track_name']
      .count()
      .reset_index(name='play_count')
)

# Total plays per year
totals = df.groupby('year')['track_name'].count().rename('year_total')

# Merge totals
top5 = top5.merge(totals, on='year')

# Compute percent
top5['pct_of_year'] = (top5['play_count'] / top5['year_total']) * 100

# Keep only top 5 each year
top5_final = (
    top5.sort_values(['year','play_count'], ascending=[True,False])
        .groupby('year')
        .head(5)
        .reset_index(drop=True)
)

top5_final
```

2020 (total plays:8106)

artist	No of plays
Pritam	671 (8.27%)
Arjit singh	630 (7.77%)
Atif aslam	288 (3.55%)
Chris Brown	240 (2.96%)
AR Rahman	225 (2.77%)

2021 (total plays:11948)

artist	No of plays
AR Rahman	1181 (9.88%)
Pritam	572 (4.78%)
Anirudh	564 (4.72%)
Arijit singh	459 (3.84%)
Atif aslam	275 (2.3%)

2022 (total plays:10523)

artist	No of plays
AR Rahman	969 (9.2%)
Yuvan	605 (5.7%)
Pritam	534 (5%)
Anirudh	451 (4.2%)
Chris Brown	354 (3.36%)

2023 (total plays:9370)

artist	No of plays
AR Rahman	461 (4.9%)
Pritam	347 (3.7%)
Chris Brown	310 (3.3%)
Anirudh	307 (3.27%)
Yuvan	271 (2.89%)

2024 (total plays:7249)

artist	No of plays
Pritam	438 (6%)
Chris Brown	317 (4.37%)
The Weeknd	304 (4.19%)
AR Rahman	299 (4.12%)
Arijit singh	293 (4%)

2025 (total plays:9125)

artist	No of plays
Drake	384 (4.2%)
Chris Brown	345 (3.78%)
Usher	170 (1.86%)
Metro Boomin	145 (1.58%)
The Weeknd	145 (1.58%)

Song Diversity Score

Diversity Score = Unique Songs / Total Plays

```
simple_diversity = (
    df.groupby('year')
     .agg(
        total_minutes=('min_played','sum'),
        total_plays=('track_name','count'),
        unique_songs=('track_name','nunique')
     )
)

simple_diversity['diversity_score'] = (
    simple_diversity['unique_songs'] / simple_diversity['total_plays']
)

print(simple_diversity[['total_minutes','total_plays','diversity_score']])
```

✓ 7.2s

```
      total_minutes  total_plays  diversity_score
year
2020   25083.348000         8106         0.227362
2021   38582.831033        11948         0.234265
2022   36913.695200        10523         0.241281
2023   29971.929083         9370         0.326467
2024   26229.720700         7249         0.208305
2025   31530.039600         9125         0.268712
```

What the score means:

- **Close to 0** → you repeat songs a lot
- **Close to 1** → you rarely repeat songs
- **0.2–0.4** → normal listeners
- **0.5+** → extremely varied listening
- **0.1 or less** → you listen to the same few songs repeatedly

Year	Total Minutes	Total Plays	Diversity Score	Meaning of Diversity Score
2020	25,083.35	8,106	**0.227**	About **22.7%** of your plays were unique songs → listening was **repetitive**.
2021	38,582.83	11,948	**0.234**	Around **23.4%** unique → still **low diversity**; many repeats.
2022	36,913.70	10,523	**0.241**	**24.1%** unique → similar pattern; moderate repetition.
2023	29,971.93	9,370	**0.326**	**32.6%** unique → **highest diversity**; you explored many more songs.
2024	26,229.72	7,249	**0.208**	Only **20.8%** unique → very **repeat-heavy year**.
2025	31,530.04	9,125	**0.269**	**26.9%** unique → moderate diversity; more variety than 2020–2022.

Most consistent songs – [songs heard lot of times all the years ,ie.song is present in all the years]

```
# Step 1 — Find how many years each song appears in
song_years = df.groupby('track_name')['year'].nunique()

# Step 2 — Only keep songs that appear in ALL years
years_in_data = df['year'].nunique()
consistent_songs = song_years[song_years == years_in_data].index

# Step 3 — Filter original dataframe to only these songs
df_consistent = df[df['track_name'].isin(consistent_songs)]

# Step 4 — Compute total plays per song
consistent_song_stats = (
    df_consistent.groupby('track_name')
                 .size()
                 .reset_index(name='total_plays')
                 .sort_values('total_plays', ascending=False)
                 .head(10)
)

print(consistent_song_stats)
```

track_name	total_plays
Darkhaast	147
Do It Again	139
Five More Hours	123
Break Your Heart	117
Raabta	110
Don't Wake Me Up	108
Forever	105
Guzarish	99
Into You	95
Die For You	91

PLATFORM BASED LISTENING

```
df.groupby(['year','platform'])['min_played'].sum()
```

[164] ✓ 2.2s

```
...  year  platform
     2020  lava mobile      10071.278300
           realme mobile    14592.969283
           redmi mobile       419.100417
     2021  realme mobile    38582.831033
     2022  realme mobile    36913.695200
     2023  realme mobile    29971.929083
     2024  realme mobile    26229.720700
     2025  realme mobile    31530.039600
     Name: min_played, dtype: float64
```

We can see which all platforms where used all the years,
In 2020 – 3 platforms were used, later on only one platform was used.

Interaction-based Analysis

Skip rate by hour
Finding out skip rate by hour, that is how many skips ere made out of total no. of songs played in that hour

```
skip_by_hour = (
    df.groupby('hour')
      .agg(
          total_plays = ('skipped', 'count'),
          skipped_plays = ('skipped', 'sum')
      )
      .assign(skip_rate = lambda x: x['skipped_plays'] / x['total_plays'])
      .reset_index()
)

print(skip_by_hour)
```

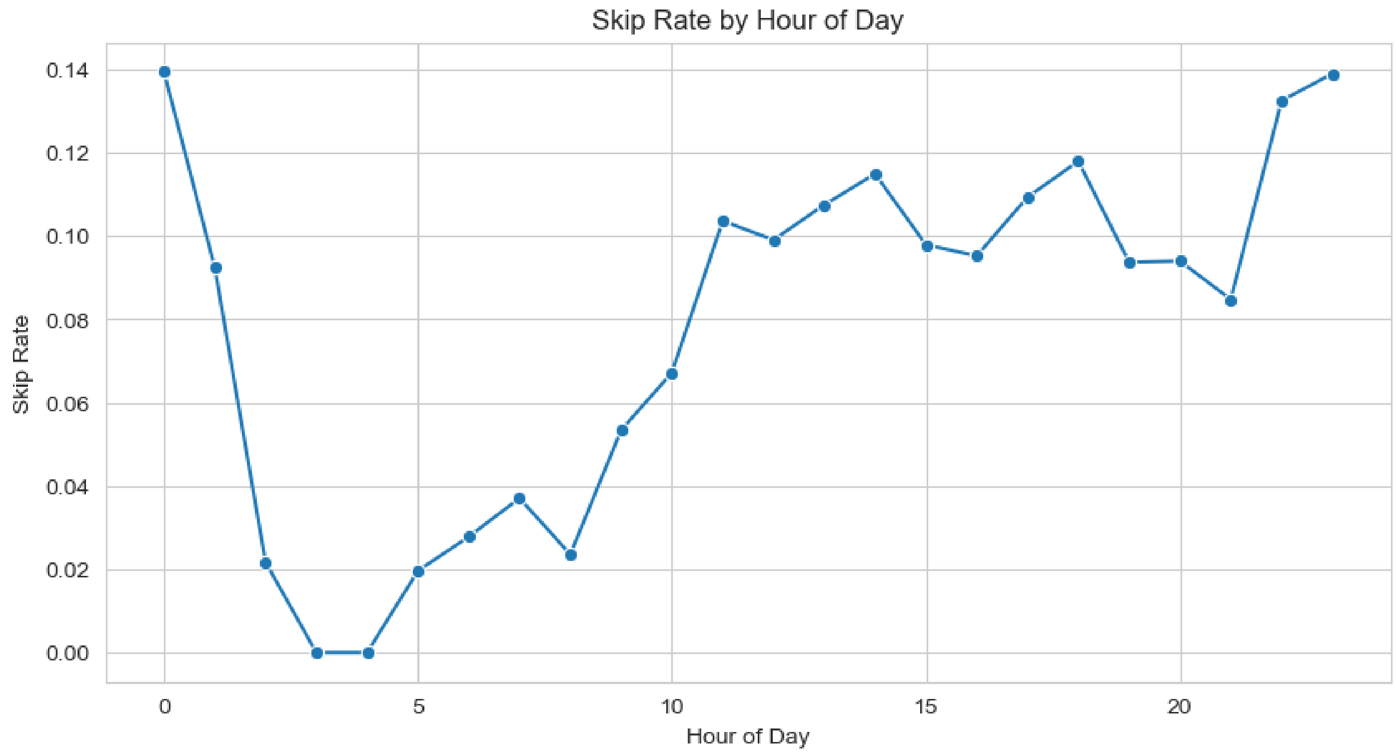
[166] ✓ 2.1s

```
   hour  total_plays  skipped_plays  skip_rate
0     0         3530            492   0.139377
1     1          833             77   0.092437
2     2          322              7   0.021739
3     3          173              0   0.000000
```

hour	total_plays	skipped_plays	skip_rate
0	3530	492	0.139377
1	833	77	0.092437
2	322	7	0.021739
3	173	0	0.000000
4	81	0	0.000000
5	51	1	0.019608
6	36	1	0.027778
7	81	3	0.037037
8	382	9	0.023560
9	675	36	0.053333
10	1221	82	0.067158
11	2434	252	0.103533
12	2940	291	0.098980
13	3166	340	0.107391
14	3397	390	0.114807
15	3119	305	0.097788
16	3416	325	0.095141
17	3140	343	0.109236
18	3149	371	0.117815
19	3846	360	0.093604
20	4334	407	0.093909
21	3849	326	0.084697
22	5600	741	0.132321
23	6554	910	0.138847

We can see more skips in the hours of 10 p.m,11 p.m,12 a.m There is 0 skip rate at the hours of 3 am to 5 am, probably they were played involuntarily.

```python
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10,5))
sns.lineplot(data=skip_by_hour, x='hour', y='skip_rate', marker='o')
plt.title("Skip Rate by Hour of Day")
plt.xlabel("Hour of Day")
plt.ylabel("Skip Rate")
plt.grid(True)
plt.show()
```



```python
never_skipped = song_stats[
    (song_stats["skips"] == 0) & (song_stats["total_plays"] >= 100)
].sort_values("total_plays", ascending=False)

never_skipped.head(20)
```

[84] ✓ 0.0s

	total_plays	skips	skip_rate
track_name			
Five More Hours	123	0	0.0
Break Your Heart	117	0	0.0

Songs that were played more than 100 times and not skipped are – "five more hours","break your heart", 100 is the threshold.

```python
song_stats = (
    df.groupby("track_name")
      .agg(
          total_plays=("track_name", "count"),
          skips=("skipped", "sum")
      )
)

song_stats["skip_rate"] = song_stats["skips"] / song_stats["total_plays"]

# minimum 20 plays
song_stats_filtered = song_stats[song_stats["total_plays"] >= 100]

most_skipped_songs = song_stats_filtered.sort_values("skip_rate", ascending=False).head(20)
most_skipped_songs
```

[91] ✓ 0.1s

	total_plays	skips	skip_rate
track_name			
Munbe Vaa	125	15	0.120000
Heartless	141	15	0.106383
Forever	105	11	0.104762
Ennodu Nee Irundhaal	100	9	0.090000

Songs played more than 100 times , and have high skip rates are – "munbe vaa","heartless","forever".

```python
# Group by year and skipped status
summary = df.groupby(['year', 'skipped']).size().unstack(fill_value=0)

# Add skip rate column
summary['skip_rate'] = summary[True] / (summary[True] + summary[False])

# Optional: reset index for nicer display
summary = summary.reset_index()

print(summary)
```

[196] ✓ 1.3s

```
skipped  year  False  True  skip_rate
0        2020   8109     0   0.000000
1        2021  11948     0   0.000000
2        2022  10038   486   0.046180
3        2023   6794  2576   0.274920
4        2024   5877  1372   0.189267
5        2025   7494  1635   0.179100
```

year	False	True	skip_rate
2020	8109	0	0.000000
2021	11948	0	0.000000
2022	10038	486	0.046180
2023	6794	2576	0.274920
2024	5877	1372	0.189267
2025	7494	1635	0.179100

There are no skips in years 2020 & 2021,
Highest skips are in 2023.

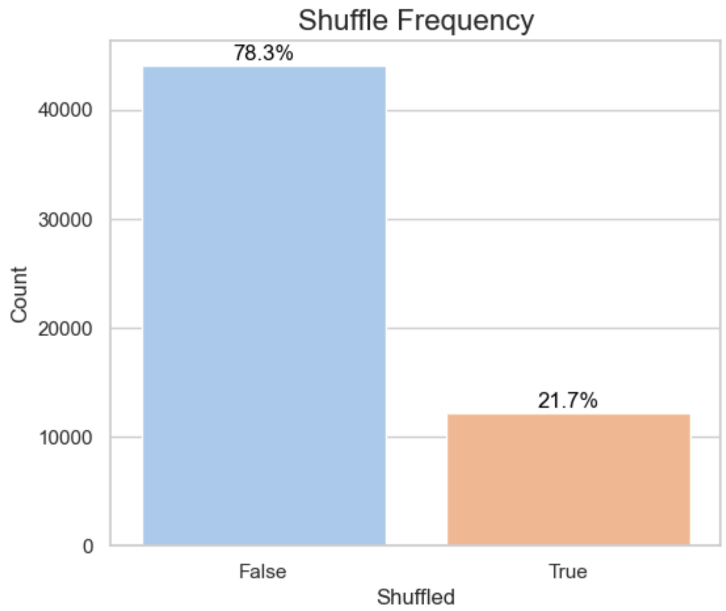
Shuffle usage

```python
import pandas as pd

# Count shuffle occurrences
shuffle_summary = df['shuffle'].value_counts().reset_index()
shuffle_summary.columns = ['shuffled', 'count']

# Calculate percentage
shuffle_summary['percentage'] = shuffle_summary['count'] / shuffle_summary['count'].sum() * 100

print(shuffle_summary)
```

[199] ✓ 0.0s

```
   shuffled  count  percentage
0     False  44123   78.330878
1      True  12206   21.669122
```

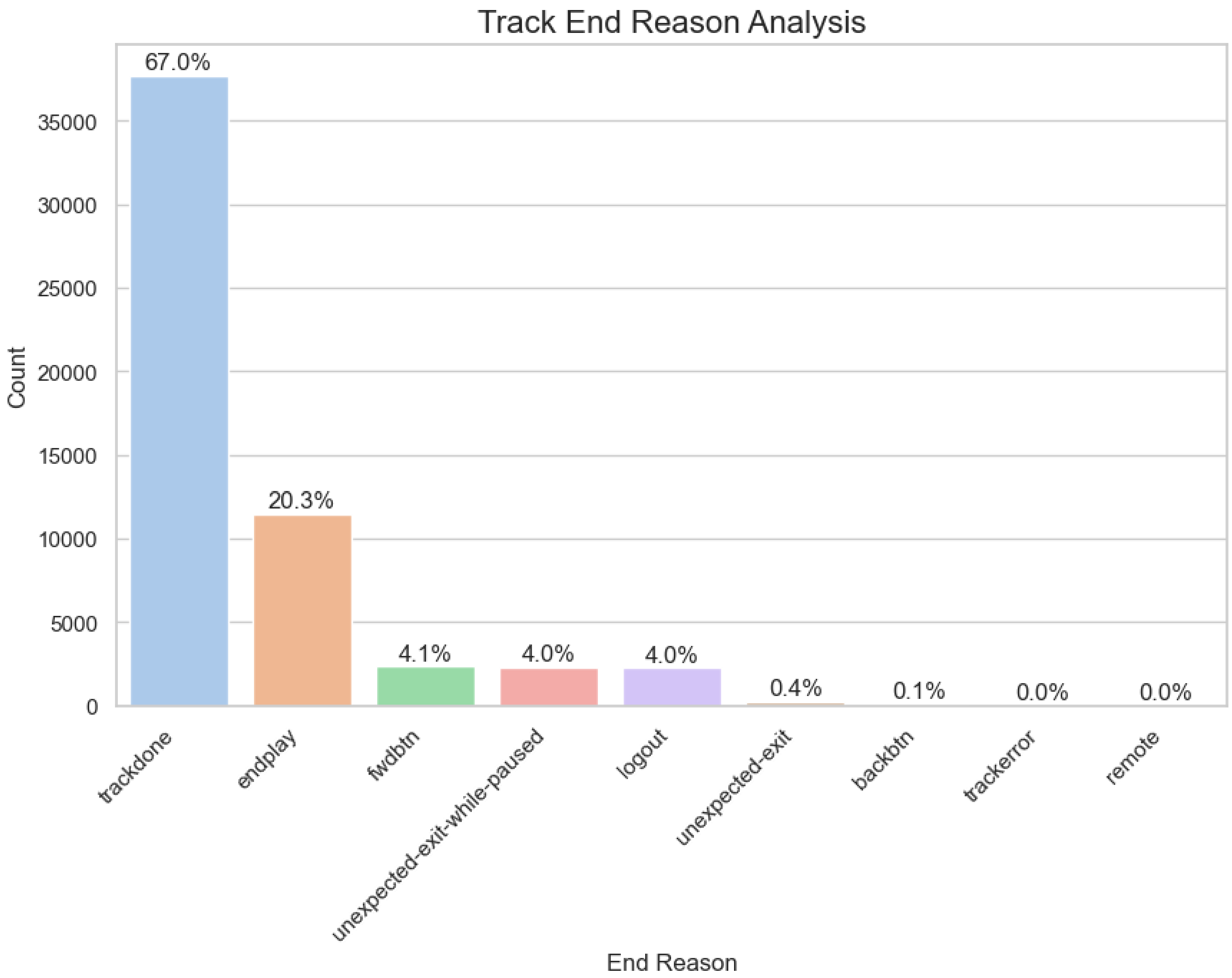
```python
# Set Seaborn style
sns.set_theme(style="whitegrid")

# Bar plot
plt.figure(figsize=(6,5))
ax = sns.barplot(
    x='shuffled',
    y='count',
    data=shuffle_summary,
    palette=sns.color_palette("pastel")
)

# Add percentages on top of bars
for i, row in shuffle_summary.iterrows():
    ax.text(i, row['count'] + max(shuffle_summary['count'])*0.01, f"{row['percentage']:.1f}%",
            color='black', ha="center", fontsize=12)

plt.title("Shuffle Frequency", fontsize=16)
plt.xlabel("Shuffled")
plt.ylabel("Count")
plt.show()
```



- True → tracks played in shuffle mode – 21.6%
- False → tracks played in normal order (manual selection)– 78.3%

Reason for the ending of track?

```python
sns.set_theme(style="whitegrid")
plt.figure(figsize=(10,6))

# Bar plot
ax = sns.barplot(x='reason', y='count', data=reason_counts, palette='pastel')

# Add percentage labels on top of bars
for i, row in reason_counts.iterrows():
    ax.text(i, row['count'] + max(reason_counts['count'])*0.01, f"{row['percentage']:.1f}%", ha='center')

plt.xticks(rotation=45, ha='right')
plt.title("Track End Reason Analysis", fontsize=16)
plt.ylabel("Count")
plt.xlabel("End Reason")
plt.show()
```



Reason	Meaning
trackdone	Song finished normally
endplay	Ended by playback system / playlist
fwdbtn	User pressed forward/next → skip song
backbtn	User pressed back → replay song
unexpected-exit-while-paused	App/device closed while paused
logout	User logged out
unexpected-exit	App/device closed unexpectedly
trackerror	Error with track
remote	Control via remote/device

clustering

```
import pandas as pd
from sklearn.cluster import KMeans

# Sum minutes played per month
monthly_listen = df.groupby(['year', 'month'])['min_played'].sum().reset_index()

# Apply K-Means clustering into 3 clusters
kmeans = KMeans(n_clusters=3, random_state=42)
monthly_listen['cluster'] = kmeans.fit_predict(monthly_listen[['min_played']])

# Calculate mean per cluster
cluster_means = monthly_listen.groupby('cluster')['min_played'].mean().sort_values()

# Map cluster number to Light/Normal/Heavy
cluster_to_label = {}
cluster_to_label[cluster_means.index[0]] = 'Light'    # cluster with lowest mean
cluster_to_label[cluster_means.index[1]] = 'Normal'   # middle mean
cluster_to_label[cluster_means.index[2]] = 'Heavy'    # highest mean

# Assign category
monthly_listen['category'] = monthly_listen['cluster'].map(cluster_to_label)

# Keep only necessary columns
monthly_listen = monthly_listen[['year','month','min_played','category']]
monthly_listen

# Group by category and calculate average minutes played
category_avg = monthly_listen.groupby('category')['min_played'].mean().reset_index()

# Optional: round for readability
category_avg['min_played'] = category_avg['min_played'].round(2)

category_avg
```

Top heavy, light and normal months of listening

Heavy listening months(16.4% of the total are heavy months with an average of 5001 mins)

Year	Month	Minutes
2020	10	4791.71
2020	11	4185.44
2020	12	5269.00
2021	1	4941.75
2021	2	4232.98
2021	3	5470.39
2022	3	4583.18
2022	9	5216.62
2025	8	5908.35
2025	9	4790.94
2025	10	5625.83

normal listening months(32.8% of the total are normal months with an average of 3171 mins)

Year	Month	Minutes Played
2020	7	2941.61
2020	9	2791.58
2021	5	2612.27
2021	6	3331.42
2021	7	2615.11
2021	8	3702.92
2021	11	2685.94
2022	1	3358.26
2022	4	2976.69
2022	8	3841.34
2022	10	3761.07
2022	11	3393.71
2022	12	2615.61
2023	1	3029.71
2023	3	3850.56
2023	4	2997.10
2023	6	3560.07
2024	3	2752.19
2024	7	3665.01
2024	10	3335.01
2025	6	2960.56
2025	7	3002.57

light listening months(50.7% are light months with an average of 1868 mins)

Year	Month	Minutes Played
2020	5	716.29
2020	6	2173.06
2020	8	2214.65
2021	4	2395.60
2021	9	2082.74
2021	10	2103.23
2021	12	2408.47
2022	2	2464.93
2022	5	1915.97
2022	6	1492.89
2022	7	1293.44
2023	2	2270.86
2023	5	2178.03
2023	7	1729.26
2023	8	2438.49
2023	9	2504.39
2023	10	2411.93
2023	11	1573.71
2023	12	1427.82
2024	1	1544.98
2024	2	1401.20
2024	4	1963.31
2024	5	1782.15
2024	6	2475.63
2024	8	1908.95
2024	9	2340.74
2024	11	2009.25
2024	12	1051.31
2025	1	1624.88
2025	2	1316.72
2025	3	547.72
2025	4	1735.53
2025	5	1842.20
2025	11	2174.74

Summary table

```python
summary = []

for y in sorted(df['year'].unique()):
    d = df[df['year'] == y]

    total_minutes = d['min_played'].sum()
    total_count = len(d)

    active_days = d['date'].nunique()
    avg_minutes_per_day = total_minutes / active_days
    avg_plays_per_day = total_count / active_days

    top_artist = (
        d.groupby('artist')['track_name']
         .count()
         .sort_values(ascending=False)
         .index[0]
    )

    top_song = (
        d.groupby('track_name')['artist']
         .count()
         .sort_values(ascending=False)
         .index[0]
    )
```

```python
    summary.append([
        y,
        total_minutes,
        total_count,
        avg_minutes_per_day,
        active_days,
        avg_plays_per_day,
        top_artist,
        top_song
    ])

columns = [
    "year", "total_minutes", "total_plays",
    "avg_minutes_per_day",
    "active_days", "avg_plays_per_day",
    "top_artist", "top_song"
]

summary_df = pd.DataFrame(summary, columns=columns)
summary_df
```

s.no	Year	Total minutes	Total plays	Avg minutes per day	Active days	Avg plays per day	Top artist	Top song
0	2020	25083.348000	8109	118.317679	212	38.250000	Pritam	Tera Yaar Hoon Main (From "Sonu Ke Titu Ki Sweety")
1	2021	38582.831033	11948	110.870204	348	34.333333	A.R. Rahman	Hosanna
2	2022	36913.695200	10524	107.307253	344	30.593023	A.R. Rahman	Pookkalae Sattru Oyivedungal
3	2023	29971.929083	9370	89.736315	334	28.053892	A.R. Rahman	Vilambara Idaiveli - From "Imaikkaa Nodigal"
4	2024	26229.720700	7249	78.767930	333	21.768769	Pritam	Heartless
5	2025	31530.039600	9129	114.654689	275	33.196364	Drake	Fire & Desire