5_pandas

October 24, 2023

1 Cleaning, Transforming and Storing your Data

Now we know a little about selecting, filtering and sorting our data, we can move on to cleaning and transforming our data too.

Data is rarely perfect, missing values, anomalous values and duplicates can cause all sorts of issues in analysis. It may also be that data is not interpreted correctly straight away, and that we need to tell Pandas what kind of data it's looking at.

Once we've got on top of that we can also explore how Pandas powerful summarisation tools can help us understand our data better.

Pandas Documentation

[ ]:
```python
import pandas as pd
filename = 'spotify_top_songs.csv'
songs_df = pd.read_csv(filename)
songs_df.head()
```

[ ]:
```
                 track_id          track_name      artists  \
0  5mjYQaktjmjcMKcUIcqz4s           Strangers  Kenya Grace
1  56y1jOTKOXSvJzVv9vHQBK  Paint The Town Red     Doja Cat
2  1reEeZH9wNt4z1ePYLyC7p              greedy   Tate McRae
3  59NraMJsLaMCVtwXTSia8i               Prada        cassö
4  5aIVCx5tnkOntmdiinnYvw               Water         Tyla

                    genre  release_year release_date  explicit  popularity  \
0  singer-songwriter pop          2023   2023-09-01     False          97
1              dance pop          2023   2023-09-20      True          87
2                  alt z          2023   2023-09-13      True          31
3             ***OOPS!***          2023   2023-08-11      True          94
4             ***OOPS!***          2023   2023-07-28     False          91

   duration_ms            playlist_name  danceability  loudness  speechiness  \
0       172964  Top 50 - United Kingdom         0.628    -8.307          NaN
1       230480  Top 50 - United Kingdom         0.864    -7.683       0.1940
2       131872  Top 50 - United Kingdom         0.750    -3.190       0.0322
3       132359  Top 50 - United Kingdom         0.638    -5.804       0.0375
4       200255  Top 50 - United Kingdom         0.673    -3.495       0.0755
```

```
  playlist_type
0     mixed_pop
1     mixed_pop
2     mixed_pop
3     mixed_pop
4     mixed_pop
```

1.1 Data Cleaning

Data cleaning can involve a range of techniques, but the unifying goal is to get your data into a state that is ready for analysis. This could include: - Removing rows where data is missing - Replacing missing data with another value. - Replacing data that may be oddly formatted to make it more analysis compatible. - Transforming the type of data in a column to correct mistakes or to make it more useful.

If we examine the `.info()` we can quickly identify if there are missing values that Pandas knows about by comparing the total entries with the 'Non-Null Count' for each column.

1.1.1 Dropping and filling missing data

[ ]:
```
songs_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1280 entries, 0 to 1279
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   track_id       1280 non-null   object
 1   track_name     1280 non-null   object
 2   artists        1280 non-null   object
 3   genre          1280 non-null   object
 4   release_year   1280 non-null   int64
 5   release_date   1280 non-null   object
 6   explicit       1280 non-null   bool
 7   popularity     1280 non-null   int64
 8   duration_ms    1280 non-null   int64
 9   playlist_name  1280 non-null   object
 10  danceability   1280 non-null   float64
 11  loudness       1280 non-null   float64
 12  speechiness    1279 non-null   float64
 13  playlist_type  1280 non-null   object
dtypes: bool(1), float64(3), int64(3), object(7)
memory usage: 131.4+ KB
```

We can identify which row is missing that value for speechiness using a special filter called `.isna()`

[ ]:
```
songs_df[songs_df['speechiness'].isna()]
```

```
[ ]:                  track_id track_name      artists                  genre  \
 0  5mjYQaktjmjcMKcUIcqz4s  Strangers  Kenya Grace  singer-songwriter pop

    release_year release_date  explicit  popularity  duration_ms  \
 0          2023   2023-09-01     False          97       172964

              playlist_name  danceability  loudness  speechiness playlist_type
 0  Top 50 - United Kingdom         0.628    -8.307          NaN     mixed_pop
```

There are multiple approaches to missing data, depending on your analysis. The simplest approach is to simply drop any rows that have any missing data. `.dropna()` will do this for us, returning a version of the dataframe where every row has a value for every column.

If you only want to drop rows with a missing value in a specific column(s) you can use the `subset=` argument. You must pass it a list of column names, even when only checking one column.

We can see that the total number of rows is now one less than the original dataframe.

```
[ ]: songs_df.dropna(subset=['speechiness']).info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1279 entries, 1 to 1279
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   track_id       1279 non-null   object
 1   track_name     1279 non-null   object
 2   artists        1279 non-null   object
 3   genre          1279 non-null   object
 4   release_year   1279 non-null   int64
 5   release_date   1279 non-null   object
 6   explicit       1279 non-null   bool
 7   popularity     1279 non-null   int64
 8   duration_ms    1279 non-null   int64
 9   playlist_name  1279 non-null   object
 10  danceability   1279 non-null   float64
 11  loudness       1279 non-null   float64
 12  speechiness    1279 non-null   float64
 13  playlist_type  1279 non-null   object
dtypes: bool(1), float64(3), int64(3), object(7)
memory usage: 141.1+ KB
```

As `.dropna()` returns a version of the dataframe without the offending rows, if we want to continue working with the cleaned version, we simply overwrite the variable with the new dataframe.

This is shown below as a comment as we don't want to actually do that just yet!

```
[ ]: # songs_df = songs_df.dropna()
```

If we wanted to keep the row, we could instead replace the missing value with another value such as the average value for that column. There are other ways to generate replacement data but they have their issues. In general it is usually better to drop these rows unless you absolutely have to keep them.

Again to save the transformed result we overwrite, but this time we overwrite the specific column in the dataframe. Shown below as a comment to avoid committing changes.

[ ]:
```
# avg_speechiness = songs_df['speechiness'].mean()
# songs_df['speechiness'] = songs_df['speechiness'].fillna(avg_speechiness)
```

1.1.2 When missing data doesn't look missing

Sometimes datasets can fool you into thinking they're more complete than they are. According to `.info()` there are no missing values in the `genre` column. However if we look at the data we can see an odd value called `***OOPS!***`. This looks like a placeholder value entered if data collection went wrong.

We can replace this with a `NaN` or `NA`, an object that represents a missing value - when we used `.isna`, `.dropna` and `.fillna` Pandas was specifically looking for these `NA` objects.

First let's check how many of these odd placeholder values we have.

[ ]:
```
songs_df[songs_df['genre'] == '***OOPS!***'].info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 50 entries, 3 to 1210
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   track_id       50 non-null     object
 1   track_name     50 non-null     object
 2   artists        50 non-null     object
 3   genre          50 non-null     object
 4   release_year   50 non-null     int64
 5   release_date   50 non-null     object
 6   explicit       50 non-null     bool
 7   popularity     50 non-null     int64
 8   duration_ms    50 non-null     int64
 9   playlist_name  50 non-null     object
 10  danceability   50 non-null     float64
 11  loudness       50 non-null     float64
 12  speechiness    50 non-null     float64
 13  playlist_type  50 non-null     object
dtypes: bool(1), float64(3), int64(3), object(7)
memory usage: 5.5+ KB
```

We can `.replace()` all these values with `NaN` objects so that we have a clearer picture of our data, and can then have the option to use our other missing data cleaning methods.

[ ]:
```
songs_df['genre'] = songs_df['genre'].replace('***OOPS!***', pd.NA)
songs_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1280 entries, 0 to 1279
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   track_id       1280 non-null   object
 1   track_name     1280 non-null   object
 2   artists        1280 non-null   object
 3   genre          1230 non-null   object
 4   release_year   1280 non-null   int64
 5   release_date   1280 non-null   object
 6   explicit       1280 non-null   bool
 7   popularity     1280 non-null   int64
 8   duration_ms    1280 non-null   int64
 9   playlist_name  1280 non-null   object
 10  danceability   1280 non-null   float64
 11  loudness       1280 non-null   float64
 12  speechiness    1279 non-null   float64
 13  playlist_type  1280 non-null   object
dtypes: bool(1), float64(3), int64(3), object(7)
memory usage: 131.4+ KB
```

Now we have a more accurate representation of our missing values let's go ahead an just drop any row with missing data using `.dropna()`.

[ ]:
```
songs_df = songs_df.dropna()
songs_df.head()
```

[ ]:
```
                 track_id          track_name       artists  \
1  56y1jOTK0XSvJzVv9vHQBK     Paint The Town Red      Doja Cat
2  1reEeZH9wNt4z1ePYLyC7p                greedy    Tate McRae
5  2FDTHlrBguDzQkp7PVj16Q              Sprinter          Dave
6  1BxfuPKGuaTgP7aM0Bbdwr          Cruel Summer  Taylor Swift
7  3vkCueOmm7xQDoJ17W1Pm3  My Love Mine All Mine        Mitski

            genre  release_year release_date  explicit  popularity  \
1       dance pop          2023   2023-09-20      True          87
2           alt z          2023   2023-09-13      True          31
5      uk hip hop          2023   2023-06-01      True          94
6             pop          2019   2019-08-23     False          99
7  brooklyn indie          2023   2023-09-15     False          93

   duration_ms            playlist_name  danceability  loudness  speechiness  \
1       230480  Top 50 - United Kingdom         0.864    -7.683       0.1940
2       131872  Top 50 - United Kingdom         0.750    -3.190       0.0322
5       229133  Top 50 - United Kingdom         0.916    -8.067       0.2410
6       178426  Top 50 - United Kingdom         0.552    -5.707       0.1570
7       137773  Top 50 - United Kingdom         0.504   -14.958       0.0321
```

```
  playlist_type
1     mixed_pop
2     mixed_pop
5     mixed_pop
6     mixed_pop
7     mixed_pop
```

Unless you want to retain the index to match back to the original data later, often it is a good idea to `.reset_index()` before continuing. We use `drop=True` to ensure the original index is not retained and just cleaned away entirely.

[ ]:
```
songs_df = songs_df.reset_index(drop=True)
```

1.1.3 Fixing Wrong data types

Sometimes either due to the way data was interpreted when Pandas loaded it, or due to the way data was created, it won't necessarily be the right type of data.

In our dataset we have a `release_year` column, and currently it is listed as an `object`

[ ]:
```
songs_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1229 entries, 0 to 1228
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   track_id       1229 non-null   object
 1   track_name     1229 non-null   object
 2   artists        1229 non-null   object
 3   genre          1229 non-null   object
 4   release_year   1229 non-null   int64
 5   release_date   1229 non-null   object
 6   explicit       1229 non-null   bool
 7   popularity     1229 non-null   int64
 8   duration_ms    1229 non-null   int64
 9   playlist_name  1229 non-null   object
 10  danceability   1229 non-null   float64
 11  loudness       1229 non-null   float64
 12  speechiness    1229 non-null   float64
 13  playlist_type  1229 non-null   object
dtypes: bool(1), float64(3), int64(3), object(7)
memory usage: 126.1+ KB
```

If we look at the `release_date` value for the first row, we can see it is actually a string, and if we ask Pandas to `.describe()` it to us it can do very little as it thinks they are just words, rather than dates.

[ ]:
```
songs_df.loc[0,'release_date']
```

[ ]:
```
'2023-09-20'
```

[ ]:
```
songs_df['release_date'].describe()
```

[ ]:
```
count           1229
unique           684
top       2023-09-08
freq              15
Name: release_date, dtype: object
```

We can recast the column as a date by using `pd.to_datetime()` which takes a column of strings and returns a column of dates.

[ ]:
```
songs_df['release_date'] = pd.to_datetime(songs_df['release_date'])
songs_df['release_date'].describe(datetime_is_numeric=True)
```

[ ]:
```
count                              1229
mean      2001-01-04 20:20:53.702196864
min                 1954-01-01 00:00:00
25%                 1983-03-23 00:00:00
50%                 2008-03-28 00:00:00
75%                 2020-05-22 00:00:00
max                 2023-10-13 00:00:00
Name: release_date, dtype: object
```

1.2 Exercises 1

Take a look at section 1 of the exercises sheet. Complete the tasks before moving on.

1.3 Data Transformations

We can use Pandas to quickly transform our data to provide us quick insights into the data that would otherwise be difficult or impossible to achieve manually. For example we can: - Count the number of times particular values are used, good for categorical data. - Use `groupby` to compare different subsets of data quickly.

1.3.1 Value Counts

A simple but powerful method for quickly summarising a column of categorical data, often string data. For example we could ask how many tracks are in the dataset per playlist, and get the answer easily using `.value_counts()`.

[ ]:
```
songs_df['playlist_name'].value_counts()
```

[ ]:
```
Hit Rewind                        100
All Out 2000s                     100
All Out 2010s                     100
```

```
All Out 90s                          100
All Out 80s                          100
All Out 60s                          100
All Out 70s                           99
All Out 50s                           99
Every Official UK Number 1 Ever       93
The Pop List                          88
alt/pop                               59
Cheesy Hits!                          50
Top 50 - United Kingdom               47
Today's Top Hits                      47
Every UK Number One: 2023             47
Name: playlist_name, dtype: int64
```

Or the most mentioned artist…

[ ]:
```
artist_counts = songs_df['artists'].value_counts().head(10)
artist_counts
```

[ ]:
```
Ed Sheeran        21
Olivia Rodrigo    17
Drake             17
Taylor Swift      15
Billie Eilish     13
The Weeknd        11
Coldplay          11
Calvin Harris     11
Lewis Capaldi     10
Rihanna           10
Name: artists, dtype: int64
```

One thing to note is that value_counts returns a Series (column), just like any other column in a dataframe. It sets the index to be the unique values from the column it is counting, in our case, artist names.

This means that if you have a particular value in mind you want to check, you can use the `.loc` to select the specific value, or for convenience omit `.loc` and just use square brackets `[]`.

[ ]:
```
artist_counts.loc['Billie Eilish']
```

[ ]:
```
13
```

We could also use this index in other operations. For example we could filter our dataset to only contain records of our top 10 most mentioned artists.

[ ]:
```
top_10_artists = artist_counts.index
top_10_artists
```

```
[ ]: Index(['Ed Sheeran', 'Olivia Rodrigo', 'Drake', 'Taylor Swift',
            'Billie Eilish', 'The Weeknd', 'Coldplay', 'Calvin Harris',
            'Lewis Capaldi', 'Rihanna'],
           dtype='object')
```

```
[ ]: top_10_filter = songs_df['artists'].isin(top_10_artists)
     top_10_artists_df = songs_df[top_10_filter]
     top_10_artists_df.head()
```

```
[ ]:                   track_id                                          track_name  \
     3   1BxfuPKGuaTgP7aM0Bbdwr                                        Cruel Summer
     5   1kuGVB7EU95pJObxwvfwKS                                             vampire
     7   2YSzYUF3jWqb9YP9VXmpjE                                  IDGAF (feat. Yeat)
     11  6wf7Yu7cxBSPrRlWeSeK0Q  What Was I Made For? [From The Motion Picture …
     13  3IX0yuEVvDbnqUwMBB3ouC                                     bad idea right?

                artists             genre  release_year release_date  explicit  \
     3     Taylor Swift               pop          2019   2019-08-23     False
     5   Olivia Rodrigo               pop          2023   2023-09-08      True
     7            Drake  canadian hip hop          2023   2023-10-06      True
     11   Billie Eilish           art pop          2023   2023-07-13     False
     13  Olivia Rodrigo               pop          2023   2023-09-08      True

         popularity  duration_ms            playlist_name  danceability  loudness  \
     3           99       178426  Top 50 - United Kingdom         0.552    -5.707
     5           95       219724  Top 50 - United Kingdom         0.511    -5.745
     7           89       260111  Top 50 - United Kingdom         0.663    -8.399
     11          96       222369  Top 50 - United Kingdom         0.444   -17.665
     13          94       184783  Top 50 - United Kingdom         0.627    -3.446

         speechiness playlist_type
     3        0.1570     mixed_pop
     5        0.0578     mixed_pop
     7        0.2710     mixed_pop
     11       0.0307     mixed_pop
     13       0.0955     mixed_pop
```

`.value_counts()` also allows us to get proportions rather than frequencies by using the argument `normalize=True`. The simplest way to interpret the numbers is as a percentage. For example `0.1` is 10%, `0.04` is 4% etc.

```
[ ]: songs_df['genre'].value_counts(normalize=True)
```

```
[ ]: pop                   0.109845
     dance pop             0.080553
     album rock            0.075671
     adult standards       0.045566
     alt z                 0.022783
```

```
                      …
electro             0.000814
dutch edm           0.000814
indie rock          0.000814
danish pop          0.000814
acoustic blues      0.000814
Name: genre, Length: 190, dtype: float64
```

1.3.2 Groupby

`.groupby` allows us to quickly seperate our dataset up into groups based on the values in one or more columns.

[ ]:
```
grouped = songs_df.groupby('playlist_name')
grouped
```

[ ]:
```
&lt;pandas.core.groupby.generic.DataFrameGroupBy object at 0x110b85e80&gt;
```

[ ]:
```
grouped.get_group('Cheesy Hits!').head()
```

[ ]:
```
                   track_id          track_name          artists  \
182  2WfaOiMkCvy7F5fcp2zZ8L          Take on Me             a-ha
183  0GjEhVFGZW8afUYGChu3Rr       Dancing Queen             ABBA
184  4kbj5MwxO1bq9wjT5g9HaA   Shut Up and Dance    WALK THE MOON
185  2kQuhkFX7uSVepCD3h29g5          Smack That             Akon
186  47BBI51FKFwOMlIiX6m8ya  I Want It That Way  Backstreet Boys

            genre  release_year release_date  explicit  popularity  \
182  new romantic          1985   1985-06-01     False          88
183      europop           1976   1976-01-01     False          86
184    dance rock          2014   2014-12-02     False          85
185     dance pop          2006   2006-01-01      True          85
186      boy band          1999   1999-05-18     False          84

     duration_ms playlist_name  danceability  loudness  speechiness  \
182       225280  Cheesy Hits!         0.573    -7.638       0.0540
183       230400  Cheesy Hits!         0.543    -6.514       0.0428
184       199080  Cheesy Hits!         0.578    -3.804       0.0619
185       212360  Cheesy Hits!         0.939    -5.171       0.0467
186       213306  Cheesy Hits!         0.689    -5.830       0.0270

    playlist_type
182     mixed_pop
183     mixed_pop
184     mixed_pop
185     mixed_pop
186     mixed_pop
```

More importantly we can apply operations to our single grouped object and have them applied to each group seperately with the results returned all together. For example, let's ask what the average popularity score is for each different playlist.

[ ]:
```
grouped.mean()['popularity'].sort_values(ascending=False)
```

[ ]:
```
playlist_name
Today's Top Hits                   90.914894
All Out 2010s                      86.780000
Top 50 - United Kingdom            86.638298
Hit Rewind                         85.110000
All Out 2000s                      82.940000
Cheesy Hits!                       80.140000
Every UK Number One: 2023          79.978723
All Out 90s                        79.810000
All Out 80s                        79.720000
All Out 70s                        78.787879
All Out 60s                        74.210000
The Pop List                       72.954545
All Out 50s                        62.363636
Every Official UK Number 1 Ever    58.655914
alt/pop                            55.203390
Name: popularity, dtype: float64
```

We can use different aggregations by using the appropriate method such as `.mean`, `.count`, `.sum`, `.median` and `.nunique`.

[ ]:
```
grouped.nunique()[['track_name','artists']]
```

[ ]:
```
                                 track_name  artists
playlist_name
All Out 2000s                           100       70
All Out 2010s                           100       58
All Out 50s                              98       54
All Out 60s                             100       64
All Out 70s                              98       65
All Out 80s                             100       73
All Out 90s                             100       75
Cheesy Hits!                             50       48
Every Official UK Number 1 Ever          93       56
Every UK Number One: 2023                47       39
Hit Rewind                              100       70
The Pop List                             88       68
Today's Top Hits                         47       40
Top 50 - United Kingdom                  47       36
alt/pop                                  59       52
```

We can ask for different types of aggregation per column using `.agg`

[ ]:
```
aggregations = {'track_id':'count', 'artists':'nunique', 'popularity':'mean'}
grouped.agg(aggregations)
```

[ ]:
```
                                  track_id  artists  popularity
playlist_name
All Out 2000s                          100       70   82.940000
All Out 2010s                          100       58   86.780000
All Out 50s                             99       54   62.363636
All Out 60s                            100       64   74.210000
All Out 70s                             99       65   78.787879
All Out 80s                            100       73   79.720000
All Out 90s                            100       75   79.810000
Cheesy Hits!                            50       48   80.140000
Every Official UK Number 1 Ever         93       56   58.655914
Every UK Number One: 2023               47       39   79.978723
Hit Rewind                             100       70   85.110000
The Pop List                            88       68   72.954545
Today's Top Hits                        47       40   90.914894
Top 50 - United Kingdom                 47       36   86.638298
alt/pop                                 59       52   55.203390
```

Lastly, we can also group by more than one variable to break down the data further.

[ ]:
```
songs_df.groupby(['release_year','explicit']).agg(aggregations).loc[2020:]
```

[ ]:
```
                        track_id  artists  popularity
release_year explicit
2020         False            13        8   65.153846
             True             14        8   73.142857
2021         False            15        7   71.466667
             True             10        5   66.100000
2022         False            23       12   80.130435
             True              8        6   80.125000
2023         False           154      104   71.870130
             True             82       42   78.085366
```

You can also aggregate the same column twice in different ways by using a different syntax in `.agg`.

[ ]:
```
songs_df.groupby('playlist_name').agg(
    mean_popularity=('popularity','mean'),
      median_popularity=('popularity','median'),
       n_tracks=('track_id','count') )
```

[ ]:
```
                                 mean_popularity  median_popularity  n_tracks
playlist_name
All Out 2000s                          82.940000               82.5       100
All Out 2010s                          86.780000               86.0       100
All Out 50s                            62.363636               61.0        99
```

```
All Out 60s                          74.210000            73.0       100
All Out 70s                          78.787879            78.0        99
All Out 80s                          79.720000            79.0       100
All Out 90s                          79.810000            79.0       100
Cheesy Hits!                         80.140000            80.0        50
Every Official UK Number 1 Ever      58.655914            74.0        93
Every UK Number One: 2023            79.978723            82.0        47
Hit Rewind                           85.110000            85.0       100
The Pop List                         72.954545            73.0        88
Today's Top Hits                     90.914894            92.0        47
Top 50 - United Kingdom              86.638298            89.0        47
alt/pop                              55.203390            55.0        59
```

1.3.3 Storing your data

Whether you have gone through the process of cleaning your dataset, or you've produced some aggregations that you want to easily refer to later, you'll want to store your data in some way.

Whilst there are many options, the two simplest ways to store data are either to create a `.csv` file, or to use a `pickle` file.

CSV CSV files are a standard data format that are very common. They can be opened in other programmes like Microsoft Excel and are simple enough to be widely compatible.

The downside is that their simplicity means they can't store more complex types of data such as dates, meaning that you would have to do a little work upon loading the data to set the correct data types.

We've already seen the Pandas command for loading csv files (`.read_csv()`). We can create our own using `.to_csv`.

[ ]:
```
explicit_summary = songs_df.groupby('explicit').agg(aggregations)
explicit_summary
```

[ ]:
```
          track_id  artists  popularity
explicit
False         1005      514   76.289552
True           224      106   77.691964
```

[ ]:
```
explicit_summary.to_csv('explicit_summary.csv')
```

Pickle Files Gets its name from the idea of 'pickling' as in preserving something. Pickle files can store all sorts of complex data types. Unlike a CSV where data is translated into a simple text representation, pickle files store the actual dataframe object from Python. This means the data is stored and reloaded exactly as it is. the downside is that it has very little compatibility beyond being reloaded by Pandas, and often there can be problems trying to load a pickle file anywhere other than in the same place it was created.

[ ]:
```
songs_df.to_pickle('cleaned_songs_df.pkl')
```

[ ]:
```
songs_df_2 = pd.read_pickle('cleaned_songs_df.pkl')
songs_df_2.head()
```

[ ]:
```
                 track_id            track_name       artists  \
0  56y1jOTKOXSvJzVv9vHQBK     Paint The Town Red      Doja Cat
1  1reEeZH9wNt4z1ePYLyC7p                 greedy    Tate McRae
2  2FDTHlrBguDzQkp7PVj16Q               Sprinter          Dave
3  1BxfuPKGuaTgP7aMOBbdwr           Cruel Summer  Taylor Swift
4  3vkCueOmm7xQDoJ17W1Pm3  My Love Mine All Mine        Mitski

           genre  release_year release_date  explicit  popularity  \
0      dance pop          2023   2023-09-20      True          87
1          alt z          2023   2023-09-13      True          31
2     uk hip hop          2023   2023-06-01      True          94
3            pop          2019   2019-08-23     False          99
4  brooklyn indie          2023   2023-09-15     False          93

   duration_ms           playlist_name  danceability  loudness  speechiness  \
0       230480  Top 50 - United Kingdom         0.864    -7.683       0.1940
1       131872  Top 50 - United Kingdom         0.750    -3.190       0.0322
2       229133  Top 50 - United Kingdom         0.916    -8.067       0.2410
3       178426  Top 50 - United Kingdom         0.552    -5.707       0.1570
4       137773  Top 50 - United Kingdom         0.504   -14.958       0.0321

  playlist_type
0     mixed_pop
1     mixed_pop
2     mixed_pop
3     mixed_pop
4     mixed_pop
```

1.4 Exercises 2

Take a look at section 2 of the exercises sheet. Complete the tasks.

If there is time, work through the appropriate chapter of the McLevey textbook OR the recommended DataCamp course.

See Moodle for details.