HandsOn: Pandas operations

Notebook adapted from HKUST's VISLAB COMP4462 Lab Notebook

In the previous tutorial, we have learnt some basis of Python language and visualized the Pokemon dataset with Pandas and Altair.

In this tutorial, we will learn more about Pandas.

All the materials of the original tutorial are hosted on this GitHub repository.

Dataset

5 cells hidden

Explore the dataset with Pandas

```
# Import pandas

import pandas as pd


# Since the dataset is about 350MB and contains over 3 million records, it will
# take some time to load the dataset

df_daily_ranking = pd.read_csv('data.csv')


# Show the first 5 rows of the data

df_daily_ranking.head()
```

	Position	Track Name	Artist	Streams	URL	I
0	1	Reggaetón Lento (Bailemos)	CNCO	19272	https://open.spotify.com/track/3AEZUABDXNtecAO...	2 0
1	2	Chantaje	Shakira	19270	https://open.spotify.com/track/6mICuAdrwEjh6Y6...	2 0
2	3	Otra Vez (feat. J	Zion &	15761	https://open.spotify.com/track/3QwBODjSEzelZyV	2

```
# Compute the statistical summary of all the columns

df_daily_ranking.describe(include='all')
```

	Position	Track Name	Artist	Streams	
count	3.441197e+06	3440540	3440540	3.441197e+06	3
unique	NaN	18597	6628	NaN	
top	NaN	Shape of You	Ed Sheeran	NaN	https://open.spotify.com/track/7qiZfU4dY1
freq	NaN	19365	127064	NaN	
mean	9.464399e+01	NaN	NaN	5.189176e+04	
std	5.739567e+01	NaN	NaN	2.018035e+05	
min	1.000000e+00	NaN	NaN	1.001000e+03	
25%	4.500000e+01	NaN	NaN	3.322000e+03	
50%	9.200000e+01	NaN	NaN	9.227000e+03	
75%	1.430000e+02	NaN	NaN	2.965800e+04	

Clean up

See if any rows contain null

```
# Show 5 rows that contain null in any columns
# We will revisit the this selector syntax later

df_daily_ranking[df_daily_ranking.isnull().any(axis=1)].head()
```

	Position	Track Name	Artist	Streams	URL	D
39428	29	NaN	NaN	7362	https://open.spotify.com/track/3RXkboS74UYzN14...	20 07
39456	57	NaN	NaN	4426	https://open.spotify.com/track/4JAyIDXOqNM6qHu...	20 07
39463	64	NaN	NaN	4069	https://open.spotify.com/track/3bVbQvGVIe4n24A...	20 07

```
# Counting how many records have null values, for each column
```

```
df_daily_ranking.isnull().sum()
```

```
Position        0
Track Name    657
Artist        657
Streams         0
URL             8
Date            0
Region          0
dtype: int64
```

Drop the null rows

```
# Drop all the rows that contain any null values
# Inplace avoid copying the whole dataframe, computationally faster

df_daily_ranking.dropna(inplace=True)
```

Separate 'global' from the rest

In the dataset, there is a region "global", which is the sum of all the streams all over the world. It overlaps with other regions, treating it as equally as a region will cause "double count", so we separate it out.

```
df_daily_ranking_global = df_daily_ranking[df_daily_ranking['Region'] == 'global']
df_daily_ranking = df_daily_ranking[df_daily_ranking['Region'] != 'global']
```

Aggregate

Top artists of the year

```
# Count how many unique artists in the dataset

df_daily_ranking['Artist'].nunique()
```

```
6628
```

The following command has multiple components, `groupby`, `sum`, `sort_values`, column selection and `head`, we look into it one by one:

- `groupby` : This is like splitting the whole dataset by the "Artist" of each row, resulting in 6628 groups.
- `sum` : As there are multiple rows in each group of 6628 groups, this command sums up the values of each group. In our case, it sums up the "Streams" column of each row in each group.
- `sort_values` : After summing up, each group becomes one row of record, that is 6628 rows now. This command sort these 6628 rows by their value in "Streams" in descending order.
- select the "Streams" column: This is like ignoring other columns and only show us the "Streams" column alongside with the index column (which is the column used in the `groupby` command).
- `head` : Show the first 5 rows

```
# Show the top 5 artists of the year

df_daily_ranking.groupby('Artist').sum().sort_values('Streams', ascending=False)['Streams'].h
```

```
Artist
Ed Sheeran          4353885528
Drake               2285102445
The Chainsmokers    2081716050
Post Malone         1865412162
Luis Fonsi          1760377876
Name: Streams, dtype: int64
```

Top songs of the year

```
################################################################################
# TODO:                                                                        #
# Try to show the top 5 songs of the year!                                     #
################################################################################

df_daily_ranking.sort_values('Streams', ascending=False).head()

################################################################################
#                                END OF YOUR CODE                              #
################################################################################
```

	Position	Track Name	Artist	Streams	URL
792222	1	HUMBLE.	Kendrick Lamar	4068152	https://open.spotify.com/track/7KXjTSCq5nL1LoY.
818222	1	Look What You Made Me Do	Taylor Swift	3828478	https://open.spotify.com/track/2VjtYe7gpfUi2Ok.
			Kendrick		

Filter

Top song of the year in US

The square brackets of pandas have multiple usage, it is sometimes even confusing. For now, we look into using it as column selection and filtering.

If the value inside the brackets is a string, it will select the column, just like `df_daily_ranking['Region']` will return a column.

If the value inside the brackets is an array of boolean values, it will return only the `True` rows.

Let's see an example!

```
# Ignore the code below at the moment, all we care is "df_sample" now contains
# 10 rows, and 5 are in Region "us" and 5 in Region "hk"

df_sample = pd.concat([df_daily_ranking[df_daily_ranking['Region'] == 'us'].head(), df_daily_
df_sample
```

	Position	Track Name	Artist	Streams	
771622	1	Bad and Boujee (feat. Lil Uzi Vert)	Migos	1371493	https://open.spotify.com/track/4Km5HrUvYT
771623	2	Fake Love	Drake	1180074	https://open.spotify.com/track/343YBumqHu1
771624	3	Starboy	The Weeknd	1064351	https://open.spotify.com/track/5aAx2yezTd
771625	4	Closer	The Chainsmokers	1010492	https://open.spotify.com/track/7BKLCZ1jbUE
771626	5	Black Beatles	Rae Sremmurd	874289	https://open.spotify.com/track/6fujklziTHa

```
# To see how filter works, we now see how the "==" operator works

df_sample['Region'] == 'us'
```

```
771622      True
771623      True
771624      True
771625      True
771626      True
3366997    False
3366998    False
3366999    False
3367000    False
```

While the left column is the row id, the right column is the truth value of whether the row has "Region" equals "us". By passing this array into the dataframe, it will filter out only the rows marked as `True`, that is the first 5 rows.

```
# Only keep rows with "Region" equals "us"

df_sample[df_sample['Region'] == 'us']
```

	Position	Track Name	Artist	Streams	
771622	1	Bad and Boujee (feat. Lil Uzi Vert)	Migos	1371493	https://open.spotify.com/track/4Km5HrUvYTa
771623	2	Fake Love	Drake	1180074	https://open.spotify.com/track/343YBumqHu19

```
# By applying the same logic, we filter the whole dataset

df_daily_ranking_in_us = df_daily_ranking[df_daily_ranking['Region'] == 'us']

# Then, do the aggregation and find out the top songs in US

df_daily_ranking_in_us.groupby('Artist').sum().sort_values('Streams', ascending=False)['Strea
```

```
Artist
Drake             1250864578
Kendrick Lamar    1163890899
Post Malone        988811897
Lil Uzi Vert       773675118
Ed Sheeran         723507889
Name: Streams, dtype: int64
```

Top song of the year in Germany ('de') or your country

```
################################################################################
# TODO:                                                                        #
# Try to show the top 5 songs of the year in your favorite region!             #
```

```
# Try to show the top 5 songs of the year in your favorite region!                    #
##########################################################################################

# Selecting GERMANY & SPAIN -> Display Spain

df_sample = pd.concat(
    [df_daily_ranking[df_daily_ranking['Region'] == 'de'].head(),
     df_daily_ranking[df_daily_ranking['Region'] == 'es'].head()])
df_sample[df_sample['Region'] == 'es'].sort_values("Streams", ascending=False).head()

##########################################################################################
#                                  END OF YOUR CODE                                      #
##########################################################################################
```

	Position	Track Name	Artist	Streams	
2436709	1	Chantaje	Shakira	189721	https://open.spotify.com/track/6mICuAdrwEjh6Y
2436710	2	Reggaetón Lento (Bailemos)	CNCO	165291	https://open.spotify.com/track/3AEZUABDXNtecA
2436711	3	Safari	J Balvin	141575	https://open.spotify.com/track/6rQSrBHf7HlZ

Read dataset in JSON format

In order to find out who are the top artists or the top songs in Asia / North America / Europe, we need to link up the country code to continents. To do so, we need an extra dataset. You can download the `country.json` from the tutorial material repository on GitHub. Thanks to the authors of the GitHub repository annexare/Countries.

If you are using Google Colab, you should be able to see an arrow on the left edge of the window, it hides the panel. Open it and switch to the "Files" tab, you can see the files in the current directory. Click upload and upload the dataset you have downloaded.

```
# Read JSON file into dataframe

df_countries = pd.read_json('countries.json')
df_countries
```

	AD	AE	AF	AG	AI	AL	AM	AO
name	Andorra	United Arab Emirates	Afghanistan	Antigua and Barbuda	Anguilla	Albania	Armenia	Angola
native	Andorra	دولة الإمارات العربية المتحدة	افغانستان	Antigua and Barbuda	Anguilla	Shqipëria	Հայաստան	Angola
phone	376	971	93	1268	1264	355	374	244
continent	EU	AS	AS	NA	NA	EU	AS	AF
capital	Andorra la Vella	Abu Dhabi	Kabul	Saint John's	The Valley	Tirana	Yerevan	Luanda
currency	EUR	AED	AFN	XCD	XCD	ALL	AMD	AOA

```
# Flip the rows and columns

df_countries.transpose().head()
```

	name	native	phone	continent	capital	currency	languages
AD	Andorra	Andorra	376	EU	Andorra la Vella	EUR	[ca]
AE	United Arab Emirates	دولة الإمارات العربية المتحدة	971	AS	Abu Dhabi	AED	[ar]
AF	Afghanistan	افغانستان	93	AS	Kabul	AFN	[ps, uz, tk]
AG	Antigua and Barbuda	Antigua and Barbuda	1268	NA	Saint John's	XCD	[en]

```
# The argument "orient" does the exactly same thing

df_countries = pd.read_json('countries.json', orient='index')
```

Join

```
# Since the country code is in lower case, to match up, we need to make them
# upper case

df_daily_ranking['country'] = df_daily_ranking['Region'].str.upper()
```

```
# Merge the two dataframe on "country" column of df_daily_ranking and on the
# index column of df_country

df_daily_ranking_with_continent = df_daily_ranking.merge(df_countries, how='inner', left_on='


# Rename column, just as an example, no specific purpose

df_daily_ranking_with_continent = df_daily_ranking_with_continent.rename(columns={'continent'
```

Top artists of the year in North America

```
# Agggregate just the same as region

df_daily_ranking_in_na = df_daily_ranking_with_continent[df_daily_ranking_with_continent['Cor
df_daily_ranking_in_na.groupby('Artist').sum().sort_values('Streams', ascending=False)['Strea
```

```
Artist
Drake               1488701744
Kendrick Lamar      1281915448
Post Malone         1125828758
Ed Sheeran          1083913228
Lil Uzi Vert         837881663
Migos                764677867
The Chainsmokers     762650964
Future               627364617
The Weeknd           599949693
Luis Fonsi           582328513
21 Savage            524445998
Khalid               503672138
Kodak Black          482252053
Travis Scott         471269591
J Balvin             465769112
Calvin Harris        459310805
DJ Khaled            456748400
XXXTENTACION         439677187
Imagine Dragons      430717997
Big Sean             428376262
Name: Streams, dtype: int64
```

Top song of the year in Asia

```
################################################################################
# TODO:                                                                        #
# Try to show the top 5 songs of the year in Asia!                             #
################################################################################

df_daily_ranking_in_asia = df_daily_ranking_with_continent[df_daily_ranking_with_continent['C
df_daily_ranking_in_asia.groupby('Track Name').sum().sort_values('Streams', ascending=False)[
df_daily_ranking_in_asia['Track Name'].head()
```

```
################################################################################
#                                END OF YOUR CODE                              #
################################################################################
```

```
461999    Versace On The Floor
462000    Say You Won't Let Go
462001                  Closer
462002             All We Know
462003        Don't Wanna Know
Name: Track Name, dtype: object
```

Pivot Table

```
# Filter the artists

df_favorite_artist =  pd.concat([
                            df_daily_ranking_with_continent[df_daily_ranking_with_contine
                            df_daily_ranking_with_continent[df_daily_ranking_with_contine
                            df_daily_ranking_with_continent[df_daily_ranking_with_contine
                            ])
df_favorite_artist.head()
```

	Position	Track Name	Artist	Streams	URL
83	84	Hymn For The Weekend - Seeb Remix	Coldplay	2327	https://open.spotify.com/track/1OAiWI2oPmglaOi...
185	186	The Scientist	Coldplay	1285	https://open.spotify.com/track/75JFxkI2RXiU7L9...
195	196	Adventure Of A Lifetime	Coldplay	1236	https://open.spotify.com/track/69uxyAqqPIsUyTO...
		Hymn For			

```
# Pivot table, just like spreadsheet, define which attribute goes to rows, which
# to columns, values, and last but not least, how to aggregate them. Aggregation
# can be sum, max, min, mean, etc. In our case, we want to add up the streams
# counts, so we use "sum"

df_favorite_artist_streams = df_favorite_artist.pivot_table(values='Streams', index='Date', c
df_favorite_artist_streams.head()
```

Continent	AS	EU	NA	OC	SA
Date					
2017-01-01	134203.0	427319.0	274971.0	26141.0	215578.0
2017-01-02	155686.0	512328.0	318189.0	28136.0	293633.0
2017-01-03	174060.0	525427.0	370151.0	28885.0	318359.0
2017-01-04	197094.0	531836.0	353325.0	32528.0	324242.0

```
################################################################################
# TODO:                                                                        #
# Try to make a pivot table for your favorite song!                            #
################################################################################

df_favorite_my = df_favorite_artist.pivot_table(values='Streams', index='Date', columns='Arti
df_favorite_my.head()

################################################################################
#                                END OF YOUR CODE                              #
################################################################################
```

Artist	Calvin Harris	Coldplay	Drake
Date			
2017-01-01	2647101	1078212	7237607
2017-01-02	2447714	1307972	6991716
2017-01-03	2561794	1416882	7390569
2017-01-04	2557704	1439025	7364248
2017-01-05	2582562	1397947	7406643

Creating a frequency file from the SGD dataset in Hw02

Now let's see which functions of Pandas might be helpful for Hw02:
https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html

We need to create a frequency file of per-city and per-state shootings, along with gender and spatial information.

We are going to start by exploring our starting dataset 'SlateGunDeaths' and considering what our target dataset 'Frequency' might look like. Let's say 'Frequency' should give us the following columns: id name state #males #females lat long

But first, make sure you start by uploading the SGD file to the notebook --- use the Files tab in the left panel, as you did before. Do that now.

```
df_sgd = pd.read_csv('SlateGunDeaths.csv')
df_sgd.head()
```

	victimID	date	name	gender	age	ageGroup	city	state	lat	
0	1	2012-12-14	Antida Archuleta	F	20.0	3.0	Westminster	CO	39.893668	-105.(
1	2	2012-12-14	Charlotte Bacon	F	6.0	1.0	Newtown	CT	41.412323	-73.3
2	3	2012-12-14	Olivia Engel	F	6.0	1.0	Newtown	CT	41.412323	-73.3
		2012	Ana							

```
df_sgd.groupby('state').count()
```

state	victimID	date	name	gender	age	ageGroup	city	lat	lng	url
AK	33	33	30	33	32	33	33	33	33	33
AL	274	274	262	270	250	259	274	274	274	274
AR	134	134	129	133	121	133	134	134	134	134
AZ	248	248	205	248	228	245	248	248	248	248
CA	1447	1447	1315	1441	1366	1437	1447	1447	1447	1447
CO	146	146	129	145	128	142	146	146	146	146
CT	110	110	105	110	105	107	110	110	110	110
DC	105	105	98	104	98	104	105	105	105	105
DE	41	41	37	41	40	41	41	41	41	41
FL	777	777	716	769	692	754	777	777	777	777
FI	1	1	1	1	1	1	1	1	1	1
GA	390	390	343	386	345	374	390	390	390	390
HI	13	13	10	13	13	13	13	13	13	13
IA	56	56	50	56	51	55	56	56	56	56
ID	29	29	27	28	27	29	29	29	29	29
IL	615	615	605	615	606	612	615	615	615	615
IN	344	344	319	339	320	333	344	344	344	344
KS	112	112	100	111	99	109	112	112	112	112
KY	179	179	174	178	169	176	179	179	179	179
LA	424	424	401	420	392	411	424	424	424	424
MA	98	98	86	98	89	95	98	98	98	98
MD	332	332	321	330	318	328	332	332	332	332
ME	26	26	25	26	26	26	26	26	26	26
MI	491	491	295	486	431	473	491	491	491	491
MN	105	105	98	105	99	103	105	105	105	105
MO	355	355	330	351	332	347	355	355	355	355
MS	141	141	130	139	119	135	141	141	141	141
MT	32	32	29	32	30	32	32	32	32	32
NC	438	438	417	438	406	426	438	438	438	438

ND	8	8	6	8	7	7	8	8	8	8
NE	54	54	53	54	54	54	54	54	54	54
NH	14	14	13	14	13	13	14	14	14	14
NJ	278	278	264	277	267	274	278	278	278	278
NM	88	88	79	88	77	87	88	88	88	88
NV	121	121	112	121	112	118	121	121	121	121
NY	379	379	336	378	357	375	379	379	379	379
OH	496	496	470	495	470	492	496	496	496	496

The target dataset 'Frequency' should show the number of murders per city and have a column each for the number of male victims and the number of female victims. Our original dataset stores the gender data in one column. Let's start working with this gender column, and split it into two tables.

RI	25	25	25	25	25	25	25	25	25	25

```
males = df_sgd[df_sgd['gender'] == 'M']
females = df_sgd[df_sgd['gender'] == 'F']
males.head()
```

	victimID	date	name	gender	age	ageGroup	city	state	lat	ln
4	5	2012-12-14	Dylan Hockley	M	6.0	1.0	Newtown	CT	41.412323	-73.31142
7	8	2012-12-14	Jesse Lewis	M	6.0	1.0	Newtown	CT	41.412323	-73.31142
8	9	2012-12-14	James Mattioli	M	6.0	1.0	Newtown	CT	41.412323	-73.31142
10	11	2012-	Jack	M	6.0	1.0	Newtown	CT	41.412323	-73.31142

To count the number of murders per city, we can group by 'city' and count the ids. In addition, we need to convert the results back to a dataframe; the dataframe became a Series object when we included the 'victimID' column.

va	1	1	1	1	1	1	1	1	1	1

```
males_city = males.groupby(['city', 'state'])['victimID'].count().to_frame()
males_city.head()
```

		victimID
city	state	

To make the count column more explicit, let's rename it based on the gender. Notice that 'city' is not a column but the index. Apply the same to the 'females' subset.

Abbeville County	SC	1

```
males_city.columns = ['males']

females_city = females.groupby(['city', 'state'])['victimID'].count().to_frame()
females_city.columns = ['females']
```

We can merge the two datafiles by using the 'merge' function. We need to do an 'outer' join ('how' parameter) as some cities do not have murders cases for both males and females; an inner join would remove these cases. 'reset_index' brings the 'city' index back to a column form. 'fill_na' transforms the NaN (not a number) to zero. Finally, we can cast the results to integer as fillna returns a float value.

```
merged = males_city.merge(females_city, how='outer', left_on=['city', 'state'], right_index=T
merged = merged.fillna(0)
merged = merged.astype({'males': 'int16', 'females': 'int16'})
merged.head()
```

	city	state	males	females
0	Abbeville	LA	1	0
1	Abbeville	SC	2	0
2	Abbeville County	SC	1	0
3	Aberdeen	MD	1	0
4	Aberdeen	NC	3	0

To get the latitude and longitude, we can retrieve the values from the 'df_sgd' dataset and drop the duplicate rows. Then, we can merge the rows using the 'city' as key.

```
df_values = df_sgd[['city', 'state', 'lat', 'lng']]
df_values = df_values.drop_duplicates(subset=['city', 'state'], keep='first')
merged_2 = merged.merge(df_values, how='inner', on=['city', 'state'])
merged_2.head()
```

	city	state	males	females	lat	lng
0	Abbeville	LA	1	0	29.974650	-92.134292
1	Abbeville	SC	2	0	34.178172	-82.379015
2	Abbeville County	SC	1	0	34.249294	-82.473258
3	Aberdeen	MD	1	0	39.509556	-76.164120

To update the name column with the city name and state name, we need to use an additional file ('us-states-postal-code.csv') that matches the state code to the state name. We created this CSV file from the information in this page: https://www.factmonster.com/us/postal-information/state-abbreviations-and-state-postal-codes Download the file here: https://drive.google.com/file/d/1P1KtigoE5Gwd-ZiRpGRhp8x2ZJFaRFBs/view?usp=sharing

```
df_state_codes = pd.read_csv('us-states-postal-code.csv')
df_state_codes.head()
```

	State	PostalCode
0	Alabama	AL
1	Alaska	AK
2	American Samoa	AS
3	Arizona	AZ
4	Arkansas	AR

To make the Merge function work, we need to rename the columns to match the column names in our 'merged' dataset. An alternative solution here is using a dictionary (dict(zip(df_state_codes.PostalCode, df_state_codes.State))) but pandas is pretty handy.

```
df_state_codes.columns = ['state_name', 'state']
frequency = merged_2.merge(df_state_codes, on='state')
frequency.head()
```

	city	state	males	females	lat	lng	state_name
0	Abbeville	LA	1	0	29.974650	-92.134292	Louisiana
1	Algiers	LA	2	0	29.951051	-90.081089	Louisiana
2	Ascension Parish	LA	1	0	30.204658	-90.889630	Louisiana
3	Avondale	LA	3	0	29.912983	-90.203687	Louisiana
4	Baker	LA	1	0	30.588243	-91.168163	Louisiana

Now let's create the 'name' column in the frequency file by concatenating the city and the state name. The lambda function applies the operation per row.

```
frequency['city_state'] = frequency.apply (lambda row: row.city + ', ' + row.state_name, axis
# frequency.head()
frequency.to_csv(r'freq.csv')
```

We can finish the work by filtering out the required columns and renaming them. 'to_csv' saves the dataset back to a csv file.

```
final_frequency = frequency[['city_state', 'state', 'males', 'females', 'lat', 'lng']]
final_frequency.columns = ['names', 'state', 'males', 'females', 'lat', 'lng']
final_frequency.head()
```

	names	state	males	females	lat	lng
0	Abbeville, Louisiana	LA	1	0	29.974650	-92.134292
1	Algiers, Louisiana	LA	2	0	29.951051	-90.081089
2	Ascension Parish, Louisiana	LA	1	0	30.204658	-90.889630
3	Avondale, Louisiana	LA	3	0	29.912983	-90.203687
4	Baker, Louisiana	LA	1	0	30.588243	-91.168163

```
final_frequency.to_csv(r'frequency2.csv')
```

Finally, let's aggregate the data per state. We can extract the relevant columns that use the gender counts and sum them after grouping the rows by state. As a final step, we can rename the state name column to NA so that it matches the property name in the GeoJSON file from Mapstarter.

```
temp = frequency[['males', 'females', 'state', 'state_name']]
freq_by_state = temp.groupby(['state', 'state_name']).sum()
freq_by_state = freq_by_state.astype({'males': 'int16', 'females': 'int16'}).reset_index()
freq_by_state.columns = ['state', 'NAME', 'males', 'females']
freq_by_state
```

	state	NAME	males	females
0	AK	Alaska	25	8
1	AL	Alabama	228	42
2	AR	Arkansas	110	23
3	AZ	Arizona	208	40
4	CA	California	1281	160
5	CO	Colorado	106	39
6	CT	Connecticut	83	27
7	DC	Dist. of Columbia	96	8
8	DE	Delaware	33	8
9	FL	Florida	633	136
10	GA	Georgia	316	70
11	HI	Hawaii	7	6
12	IA	Iowa	47	9
13	ID	Idaho	23	5
14	IL	Illinois	552	63
15	IN	Indiana	296	43
16	KS	Kansas	90	21
17	KY	Kentucky	139	39
18	LA	Louisiana	369	51
19	MA	Massachusetts	87	11
20	MD	Maryland	295	35
21	ME	Maine	21	5
22	MI	Michigan	409	77
23	MN	Minnesota	91	14
24	MO	Missouri	292	59
25	MS	Mississippi	110	29
26	MT	Montana	31	1
27	NC	North Carolina	359	79
28	ND	North Dakota	7	1
29	NE	Nebraska	48	6

30	NH	New Hampshire	13	1
31	NJ	New Jersey	252	25
32	NM	New Mexico	69	19
33	NV	Nevada	99	22
34	NY	New York	329	49
35	OH	Ohio	421	74
36	OK	Oklahoma	148	35
37	OR	Oregon	66	19
38	PA	Pennsylvania	492	77
39	RI	Rhode Island	19	6
40	SC	South Carolina	176	59
41	SD	South Dakota	16	1
42	TN	Tennessee	237	48
43	TX	Texas	806	176
44	UT	Utah	41	13
45	VA	Virginia	234	46
46	VT	Vermont	11	4

```
freq_by_state.to_json(r'freq_by_state.json', orient='records')
```

Altair

This section is optional; do it for fun, on your own, if you are interested in Altair

```
import altair as alt


# In case you hit the error message of 5000 rows limitation, you can run the
# following command to disable the limitation

# alt.data_transformers.enable('default', max_rows=None)


# Same "groupby" trick we used before. The attributes used for "groupby" will
# becomes index of the dataframe, which are no longer normal columns and cannot
# be used directly. The method "reset_index" pulls out the attributes used for
# "groupby" back to normal columns

df_chart = df_favorite_artist.groupby(['Date', 'Continent']).sum()['Streams'].reset_index()
df_chart.head(10)
```

	Date	Continent	Streams
0	2017-01-01	AS	531877
1	2017-01-01	EU	2977119
2	2017-01-01	NA	5966385
3	2017-01-01	OC	371243
4	2017-01-01	SA	1116296
5	2017-01-02	AS	599896
6	2017-01-02	EU	3376245
7	2017-01-02	NA	5097489
8	2017-01-02	OC	428009
9	2017-01-02	SA	1245763

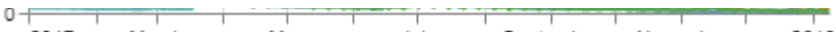
Plot dataframe as line chart. The Altair API is designed in visualization language. We can apply the knowledge learned in lectures, tell the library what we want to encode with which encoding channels and the library will handle the rest of the heavy lifting.

There are a lot more capabilities provided by the library, explore the documentation for more marks and encodings channels.

```
# State what we want the X-axis to encode, and what for the Y-axis, finally,
# what for the color channel
# Data types ("Q": quantitative, "O": ordinal, "N": nominal, "T":
# temporal) are stated with the delimiter ":" after the column name
# See documentation for more details

alt.Chart(df_chart).mark_line().encode(
    x='Date:T',
    y='Streams:Q',
    color='Continent:N'
)
```

Juxtaposition (side-by-side)

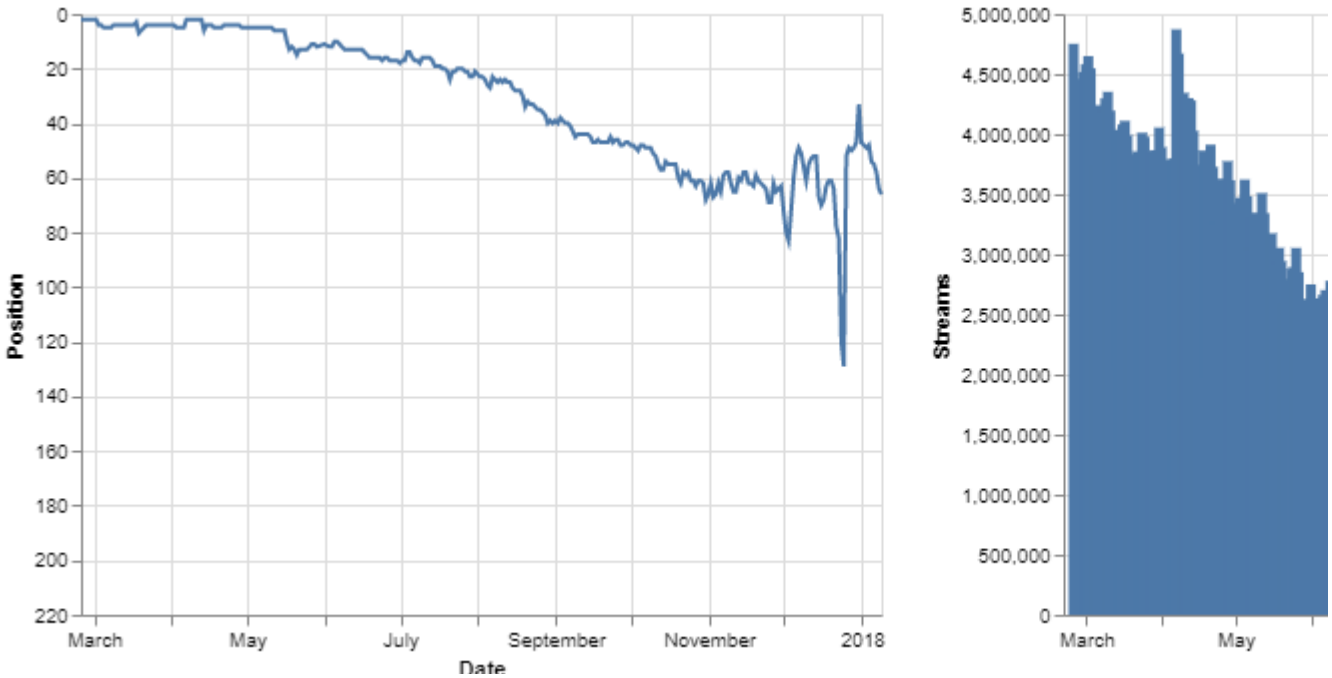
Altair makes it very easy to put multiple charts together, see the documentation for more examples and how to use this powerful feature.



```
# Filter out the song we want to visualize

df_chart = df_daily_ranking_global[df_daily_ranking_global['Track Name'] == 'Something Just l
df_chart.head()
```

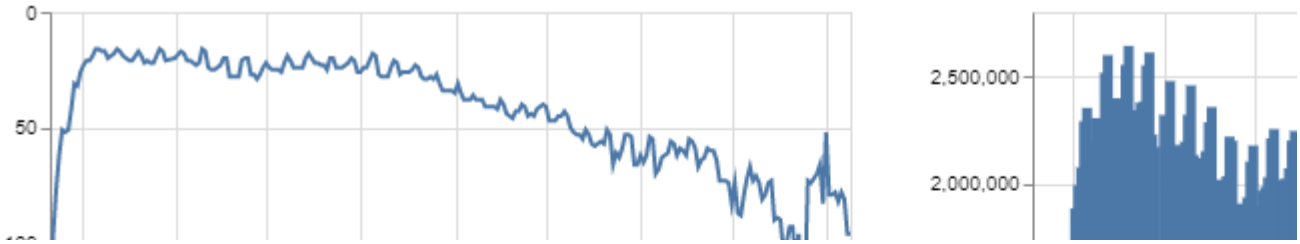
	Position	Track Name	Artist	Streams	
3127244	2	Something Just Like This	The Chainsmokers	4752225	https://open.spotify.com/track/6RUKPb4LE
3127444	2	Something Just Like This	The Chainsmokers	4460815	https://open.spotify.com/track/6RUKPb4LE
		Something			

```
# We store the return values in variables, and then we can use "|", "&" and "+"
# to combine them as horizontally, vertically or overlay, see documentation for
# more details

position_chart = alt.Chart(df_chart).mark_line().encode(
    x='Date:T',
    y=alt.Y('Position:Q', scale=alt.Scale(zero=False, padding=1, domain=[200, 1]))
)

streams_chart = alt.Chart(df_chart).mark_bar().encode(
    x='Date:T',
    y='Streams:Q'
)

position_chart | streams_chart
```




```
##########################################################################################
# TODO:                                                                                  #
# Try to visualize the trend of your favorite song!                                      #
##########################################################################################
df_chart = df_daily_ranking_global[df_daily_ranking_global['Track Name'] == 'Felices los 4']
position_chart = alt.Chart(df_chart).mark_line().encode(
x='Date:T',
y=alt.Y('Position:Q', scale=alt.Scale(zero=False, padding=1, domain=[250, 1]))
)
streams_chart = alt.Chart(df_chart).mark_bar().encode(
x='Date:T',
y='Streams:Q'
)
position_chart | streams_chart


##########################################################################################
#                                    END OF YOUR CODE                                    #
##########################################################################################
```

```
selection = alt.selection_multi(fields=['Track Name'])
color = alt.condition(selection,
                      alt.Color('Track Name:N', legend=None),
                      alt.value('lightgray'))
chart = alt.Chart(df_chart).mark_line().encode(
    x='Date:T',
    y='Streams:Q',
    color=color,
    tooltip=['Track Name', 'Streams', 'Date']
)

legend = alt.Chart(df_chart).mark_square().encode(
    y=alt.Y('Track Name:N', axis=alt.Axis(orient='right')),
    color=color
).add_selection(
    selection
)

chart | legend
```



Interactions

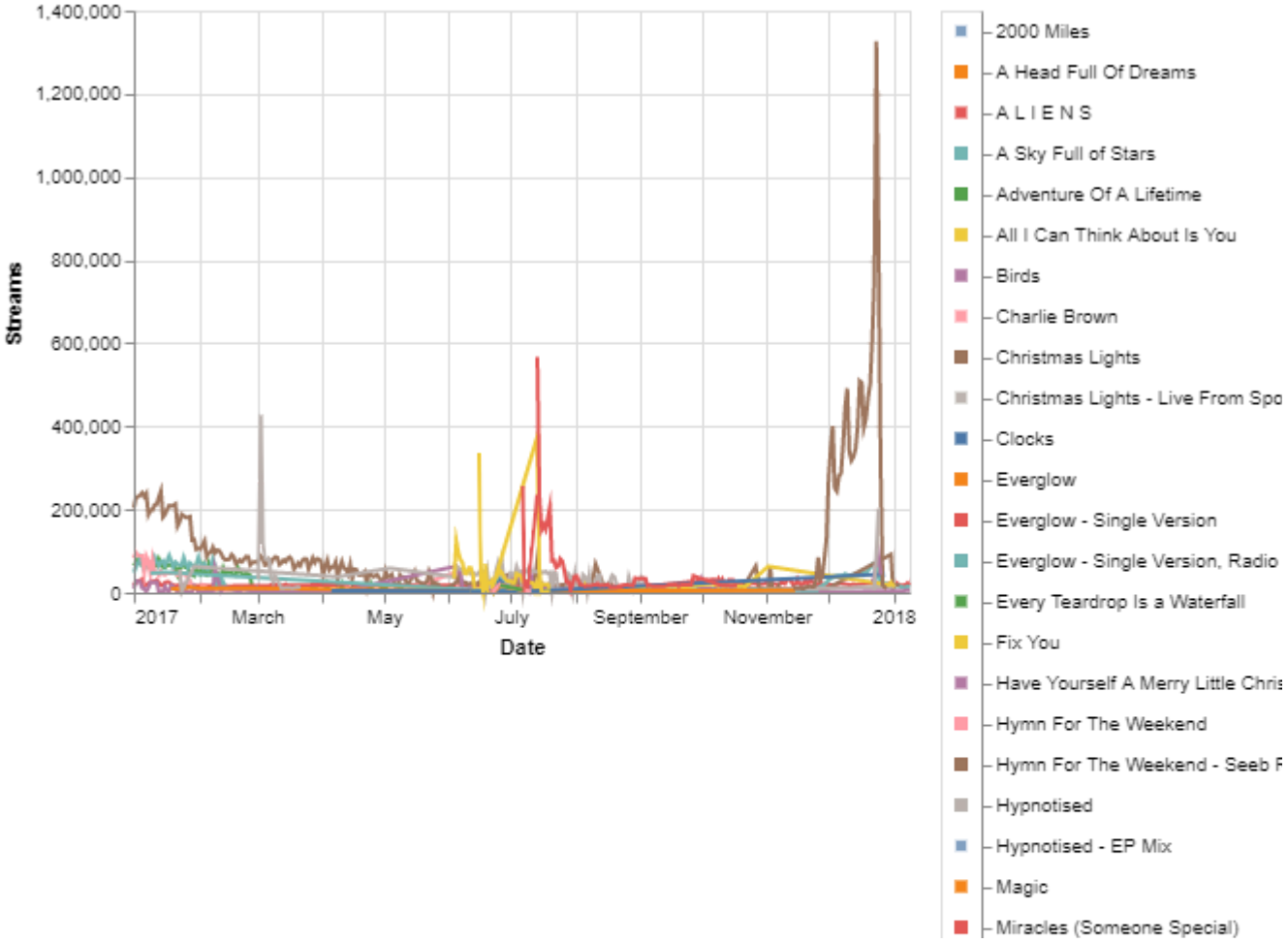
Interaction is very useful for exploring datasets. It was time consuming to make charts interactive, thanks to Altair (and the vega-lite behind the scene), it is much easier now. We will go through two examples, but there are more in the documentation, check it out!

```
# Again, filter, groupby, aggregate, reset_index, apply the tricks we have
# learnt in this tutorial

df_chart = df_favorite_artist[df_favorite_artist['Continent'] == 'EU'].groupby(['Date', 'Trac
df_chart.head()
```

	Date	Track Name	Streams
0	2017-01-01	Adventure Of A Lifetime	64682
1	2017-01-01	Controlla	138141
2	2017-01-01	Everglow - Single Version	17622
3	2017-01-01	Everglow - Single Version, Radio Edit	48726
4	2017-01-01	Fake Love	464041

```
# Add a scale interaction to allow pan and zoom. Use your scroll wheel to zoom
# and drag-and-drop to pan. More interaction techniques in documentation

scales = alt.selection_interval(bind='scales')
alt.Chart(df_chart).mark_line().encode(
    x='Date:T',
    y='Streams:Q',
    color='Track Name:N'
).add_selection(
    scales
)
```

```
# Interactively focus on data of interest (gray-out other data). We need to make
# a chart, and a selector ponel. The recommended way is to make two charts,
# putting them side-by-side, just like the juxtaposition example above. Try to
# click on the squares in the lengend, it acts like a highlighter.
```


1,400,000
1,200,000
1,000,000
800,000
600,000
400,000
200,000
0
Streams
2017
March
May
July
September
November
2018
Date
2000 Miles
A Head Full Of Dreams
A L I E N S
A Sky Full of Stars
Adventure Of A Lifetime
All I Can Think About Is You
Birds
Charlie Brown
Christmas Lights
Christmas Lights - Live From Spo
Clocks
Everglow
Everglow - Single Version
Everglow - Single Version, Radio
Every Teardrop Is a Waterfall
Fix You
Have Yourself A Merry Little Chris
Hymn For The Weekend
Hymn For The Weekend - Seeb F
Hypnotised
Hypnotised - EP Mix
Magic
Miracles (Someone Special)