1 #tbt [18 Pts]

Angela loves celebrating Throwback Thursday, a weekly tradition of posting old pictures on social media. She records all of her previous social media posts in a `DataFrame` called `pics`. The first 5 rows of `pics` and its column descriptions are given below:

- `post_id`: Unique number for each post, assigned in chronological order (type = `numpy.int64`).
- `date`: The date the picture was posted. Assume only one picture can be posted per day (type = `pandas.Timestamp`).
- `likes`: Number of likes the picture received so far (type = `numpy.int64`).
- `day_of_week`: 1 for Monday, 2 for Tuesday, etc. (type = `numpy.int64`).

	post_id	date	likes	day_of_week
0	1	2024-04-25	120	4
1	2	2024-05-02	75	4
2	3	2024-05-08	103	3
3	4	2024-05-09	84	4
4	5	2024-05-11	95	6

`pics.head()`

(a) [2 Pts] **For this part only:** If 30% of the rows in `pics` have missing values in the `date` column, what is the **BEST** option for dealing with these missing entries?

- ◯ A. Drop all rows with missing values.
- ◯ B. Impute with the mode of the `date` column.
- ◯ **C. Interpolate values using information from the rest of the `DataFrame`.**
- ◯ D. Leave the `DataFrame` as is.

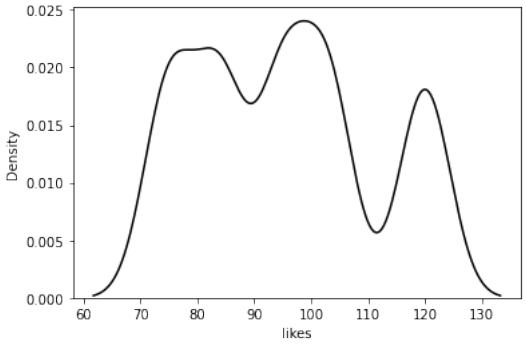
Solution: We know that each `post_id` is assigned sequentially, so we can make a good estimate for our missing dates by looking at that column.

(b) [1 Pt] **In one sentence or less:** What is the granularity of `pics`?

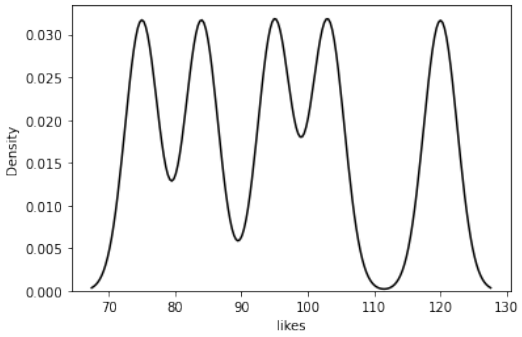
Solution: Each row represents one post/picture.

(c) [2 Pts] Angela generates the following KDE curves using a Gaussian kernel. Which KDE curve has the **smallest** bandwidth parameter?

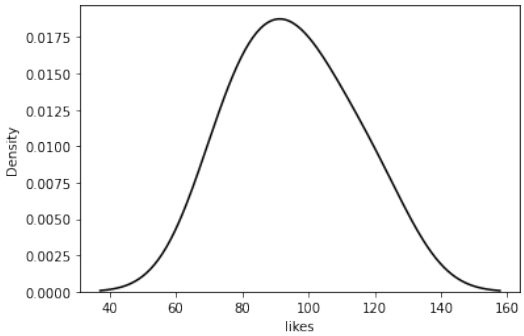
A.



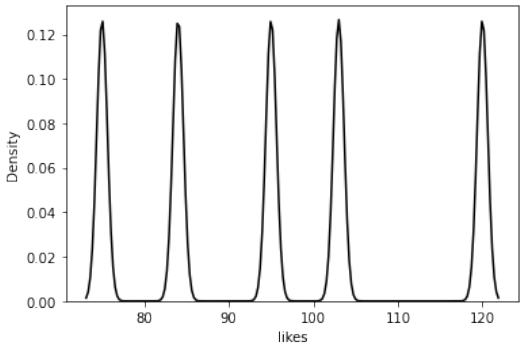
B.



C.



D.



◯ A ◯ C

◯ B ◯ **D**

Solution: D. The smoother the curve, the higher the bandwidth parameter.

Angela has a second `DataFrame` called `comments` which contains all the comments left on her pictures and the users who wrote them. The first 5 rows can be seen below:

	post_id	comment_user	comment_text
0	1	swei	lol
1	1	yashdave	lol lol :smile_cat:
2	4	lillian	looks fun!
3	4	swei	nice pic! :smile:
4	5	lillian	spotted :open_mouth: :camera:

`comments.head()`

(d) [3 Pts] In each `comment_text`, emojis are represented by combinations of alphabet characters and underscores (`"_"`) in between two colons (i.e., `":laughing_face:"`).

Angela creates a RegEx pattern called `emoji_pattern`, which she uses in the following code snippet to generate the given output:

```
example_comment = comments.iloc[4]["comment_text"]

emoji_pattern = ___________A___________

_______B_______(emoji_pattern, example_comment)
```

```
['open_mouth', 'camera']
```

(i) Fill in the blank `A` by choosing the RegEx patterns which could be `emoji_pattern`. **Select all that apply.**

- ☐ **A.** `r":([a-zA-Z_]+):"`
- ☐ B. `r":(.*):"`
- ☐ C. `r":(\w)+:"`
- ☐ **D.** `r":(\w+):"`

Solution:
Option A is correct. This allows for any alphabetical character or underscore to appear at least once.

Option B is incorrect. The `.` symbol represents any character, which would lead

to all the emojis being captured in one string rather than separately.

Option C is incorrect. The + is outside the capturing group, capturing only one character from each emoji.

Option D is correct. This allows for any combination of at least 1 letter and/or underscores.

(ii) Fill in the blank `B`:

- ◯ A. `re.match`
- ◯ B. `re.search`
- ◯ **C.** `re.findall`
- ◯ D. `str.extract`

Solution: We want the output to be a list of all individual matches, so C is the correct option.

Angela uses `pd.merge` to join `pics` and `comments` on the `post_id` column into a new `DataFrame` called `merged`. The first few rows are shown below:

	post_id	date	likes	day_of_week	comment_user	comment_text
0	1	2024-04-25	120	4	swei	lol
1	1	2024-04-25	120	4	yashdave	lol lol :smile_cat:
2	2	2024-05-02	75	4	NaN	NaN
3	3	2024-05-08	103	3	NaN	NaN
4	4	2024-05-09	84	4	lillian	looks fun!
5	4	2024-05-09	84	4	swei	nice pic! :smile:
6	5	2024-05-11	95	6	lillian	spotted :open_mouth: :camera:

```
merged.head(7)
```

(e) [2 Pts] If `pics` was the left table and `comments` was the right table, what kind of join did Angela use? Assume that there are no missing values in `comments`.

◯ A. Inner Join ◯ C. Right Join

◯ **B. Left Join** ◯ **D. Not Enough Information**

Solution: Rows from the left `DataFrame` (`pics`) which do not have a match in the right `DataFrame` (`comments`) still appear in `merged`, with the columns from `comments` filled with null values.

Note: the original solution was intended just to be a left join, but technically a full join would yield this result as well, resulting in Not Enough Information being correct as well.

(f) [3 Pts] Using `merged`, write a Python statement that creates a `DataFrame` displaying the **number of times** each `comment_user` commented on each `day_of_week`, including rows with `NaN` values. Each **column** should represent one `day_of_week` and each **row** should represent one `comment_user`. If a `comment_user` has never commented on a certain `day_of_week` **there should be a value of 0.**

Solution:
```
merged.pivot_table(index = "comment_user",
                   columns = "day_of_week",
                   aggfunc = "size",
                   fill_value = 0)
```

Nothing is needed for the `values` argument, but any column in `merged` would also work there.

The first few rows of `merged` are shown again here for your convenience:

	post_id	date	likes	day_of_week	comment_user	comment_text
0	1	2024-04-25	120	4	swei	lol
1	1	2024-04-25	120	4	yashdave	lol lol :smile_cat:
2	2	2024-05-02	75	4	NaN	NaN
3	3	2024-05-08	103	3	NaN	NaN
4	4	2024-05-09	84	4	lillian	looks fun!
5	4	2024-05-09	84	4	swei	nice pic! :smile:
6	5	2024-05-11	95	6	lillian	spotted :open_mouth: :camera:

`merged.head(7)`

(g) [5 Pts] Fill in the blanks below to create a `DataFrame` which displays the **longest** `comment_text` for each `comment_user`, as well as the **length** of these comments. Do not worry about ties.

```
merged["length"] = _____A_____
merged._____B_____._____C_____[["length", "comment_text"]]._____D_____
```

(i) Fill in blank `A`:

> **Solution:** `merged["comment_text"].str.len()`

(ii) Fill in blank `B`:

> **Solution:** `sort_values("length")`
>
> students can use `ascending = False` if they aggregate using `.agg("first")`.

(iii) Fill in blank `C`:

> **Solution:** `groupby("comment_user")`

(iv) Fill in blank `D`:

> **Solution:** `agg("last")` or `.last()`
> or `.agg("first")`, `.first()` if `.sort_values()` was used in descending order.