sp25_stat107_vgohe2 / lab_gpa / **lab_gpa.ipynb**

VedantCars2004 completed lab gpa try 1 6fd70fa · 4 days ago

2317 lines (2317 loc) · 129 KB

Welcome to Lab: GPA 🎓

In this lab, you'll explore the GPA dataset again to find out more about the courses here at UIUC!

Fun fact: the dataset you're using in this lab is the same dataset that Professor Wade uses to make the GPA visualizations! :) Check this out here if you haven't seen it already:
https://waf.cs.illinois.edu/discovery/grade_disparity_between_sections_at_uiuc/

A few tips to remember:

- **You are not alone on your journey in learning programming!** You have your lab TA, your CAs, your lab group, and the professors (Prof. Wade and Prof. Karle), who are all here to help you out!
- If you find yourself stuck for more than a few minutes, ask a neighbor or course staff for help! When you are giving help to your neighbor, explain the **idea and approach** to the problem without sharing the answer itself so they can have the same ***ah-hah*** moment!
- We are here to help you! Don't feel embarrassed or shy to ask us for help!

Let's get started!

In [276…

```
# Meet your CAs and TA if you haven't already!
# ...first name is enough, we'll know who they are! :)
ta_name = "Mallory"
ca1_name = "Urvi"
ca2_name = "Jonny"
ca3_name = ""


# Say hello to each other!
# - Groups of 3 are ideal :)
# - However, groups of 2 or 4 are fine too!
#
# Question of the Day (QOTD) to Ask Your Group: "What's your favorite soci
partner1_name = "Steven Yuan"
partner1_netid = "steveny3"
partner1_favsocialmedia = "bilibili"

partner2_name = "Brandon Cha"
partner2_netid = "bcha4"
partner2_favsocialmedia = "instagran"

partner3_name = "Eric Tong"
partner3_netid = "etong4"
partner3_favsocialmedia= "Instagram"

# Which lab section are you in? Record it below! It should start with a Y
# 2 numbers after the Y. If you're unsure - ask your TA/CAs!
```

```
lab_section = "Y15"
```

Part 1: Exploring GPA

Load the GPA Dataset

Before we begin exploring the GPA Dataset, we've got to load it in! The most recent version of the "GPA Dataset" (up to Winter 2024) is available here:

```
https://waf.cs.illinois.edu/discovery/gpa.csv
```

Use Python to load this dataset into a DataFrame called `df`:

In [277…

```
import pandas as pd
df = pd.read_csv("https://waf.cs.illinois.edu/discovery/gpa.csv")
```

🔬 Test Case Checkpoint 🔬

In [278…

```
## == CHECKPOINT TEST CASES ==
# - This read-only cell contains test cases for your previous cell.
# - If this cell runs without any errors, you PASSED all test cases!
# - If this cell results in any errors, check your previous cell, make cha
assert(len(df) == 69112), "This is not the GPA dataset you're looking for

## == SUCCESS MESSAGE ==
# You will only see this message (with the emoji showing) if you passed a
tada = "\N{PARTY POPPER}"
print(f"{tada} All tests passed! {tada}")
```

```
🎉 All tests passed! 🎉
```

Puzzle 1.1: The "Average GPA" Column

Each row in the GPA Dataset represents a **course section** at Illinois. For our exploratory data analysis, we are going to need an additional `Average GPA` column.

To create this column, we need to compute the **weighted average GPA** for each course section by taking into account the **number of students** who received each letter grade and the weight of each letter grade (see page 27 of your notebook for details). To find the grade points for each letter grade, see the Illinois Registrar.

Q1: Before creating the `Average GPA` column, talk to your group about how exactly you are going to go about doing this. What variables will you use and how will you use them? Describe the process without writing the code in the cell below.

We are going to need an integer average gpa variable to store the calculations of the average gpa. We can do this by multiplying the number of students with the weightage of each grade point, and then divide by the number of students. Then we can use python syntax to create a new column and store the variable inside it.

Using the cell below, create the `Average GPA` column in our DataFrame, `df`.

In [279…
```
average_gpa = (4 * (df["A+"]) + 4 * (df["A"]) + 3.67 * (df["A-"]) + 3.33 *
df["Average GPA"] = average_gpa
```

🔬 Test Case Checkpoint 🔬

In [280…
```
## == TEST CASES for Puzzle 1.1 ==
# - This read-only cell contains test cases for your previous cell.
# - If this cell runs without any error or output, you PASSED all test cas
# - If this cell results in any errors, check your previous cell, make cha
import math
assert( len(df) == 69112  ), "You shouldn't be changing the length of `df`
assert( "Average GPA" in df.columns), "Make sure your new column is named
assert( math.isclose(df['Average GPA'].mean(), 3.3790213685614776)), "You

## == SUCCESS MESSAGE ==
# You'll only see this message (With the emoji showing) if you passed all
tada = "\N{PARTY POPPER}"
print(f"{tada} All tests passed! {tada}")
```

🎉 All tests passed! 🎉

Puzzle 1.2: The "Hardest" and "Easiest" Courses?

One way to judge a course's difficulty is to consider its **Average GPA**.

Using the `Average GPA` column you created and the two cells below, find:

- The **50** courses with the **lowest** `Average GPA`, storing in the DataFrame `df_hard`
- The **50** courses with the **highest** `Average GPA`, storing the DataFrame `df_easy`

In [281…
```
df_hard = df.nsmallest(50, "Average GPA")
```

In [282…
```
df_easy = df.nlargest(50, "Average GPA")
```

Now, using your two new DataFrames (`df_hard` and `df_easy`), find:

- The **mean course number** of the 50 hardest courses by GPA, storing in the variable `hard_avg`
- The **mean course number** of the 50 easiest courses by GPA, storing in the variable `easy_avg`

variable `easy_avg`

In [283…
```
hard_avg = df_hard["Number"].mean()
hard_avg
```

Out[283… `np.float64(167.22)`

In [284…
```
easy_avg = df_easy["Number"].mean()
easy_avg
```

Out[284… `np.float64(372.96)`

🔬 Test Case Checkpoint 🔬

In [285…
```
## == TEST CASE for Puzzle 1.2 ==
# - This read-only cell contains test cases for your previous cell.
# - If this cell runs without any error or output, you PASSED all test cas
# - If this cell results in any errors, check your previous cell(s), make
import math
assert( len(df_hard) == len(df_easy) == 50 ), "Your df_hard and df_easy sh
assert( math.isclose(df_hard['Average GPA'].sum(), 80.3858156565443) ), "Y
assert( math.isclose(df_easy['Average GPA'].sum(), 199.39636832538582) ),
assert( math.isclose(hard_avg, 167.22) ), "Your calculation for the averag
assert( math.isclose(easy_avg, 372.96) or math.isclose(easy_avg, 375.56))

## == SUCCESS MESSAGE ==
# You'll only see this message (With the emoji showing) if you passed all
tada = "\N{PARTY POPPER}"
print(f"{tada} All tests passed! {tada}")
```

🎉 All tests passed! 🎉

Analysis: "Hardest" and "Easiest" Courses?

Q2: After solving Puzzle 1.2, your friend has the following claim:

> *"We know that the undergraduate courses are coded from 001 to 499, where a larger number (in the hundreds place) usually implies more advanced material. Based on our results in the previous puzzles, the data shows that the junior-level and senior-level courses are clearly not the hardest courses at UIUC."*

Comment on your friend's claim below. Do you think they are correct? Explain why or why not.

Yes, I would agree with this student because we just calculated the average gpa and found the mean course number of the hardest and easiest classes. Going off of these statistics, it does seem like the underclassmen courses might be more challenging due to professors, or the fact that students are still adjusting to university life. The hardest classes' average course number was about 167 while the easiest classes' average

course number was about 372, which shows that it was harder to get a good GPA in the 100 and 200 level courses.

Part 2: GPA By Subject

We've explored some of the GPA Dataset as a whole, but what if we want to investigate **differences in GPA by subject**?

Puzzle 2.1: Exploring Different Subjects

In the following cell, create a new DataFrame called `df_subject` that has a single row for each subject. In `df_subject`, each **letter grade column** should contain the **total number of students** receiving the same grade in that `Subject`. Make sure your DataFrame only contains numeric columns.

In [286…

```
df_subject = df.groupby("Subject").agg("sum", numeric_only = True)
```

🔬 Test Case Checkpoint 🔬

In [287…

```
## == TEST CASE for Puzzle 2.1 ==
# - This read-only cell contains test cases for your previous cell.
# - If this cell runs without any error or output, you PASSED all test cas
# - If this cell results in any errors, check your previous cell(s), make
import math
assert( 'df_subject' in vars() ), "Make sure your DataFrame grouped by 'Su
assert( len(df_subject) == 174 ), "Make sure you are grouping by 'Subject
assert( math.isclose(df_subject.Students.mean(), 23685.287356321838) ), "D

## == SUCCESS MESSAGE ==
# You'll only see this message (With the emoji showing) if you passed all
tada = "\N{PARTY POPPER}"
print(f"{tada} All tests passed! {tada}")
```

```
🎉 All tests passed! 🎉
```

Puzzle 2.2: Fixing our `Average GPA` Column

Your intuition may tell you that some columns in `df_subject` look off. This is correct - given the way we've grouped the data to find **total student counts** by grade, the `Year` and `Average GPA` columns are incorrect.

Let's fix this by redefining the `Average GPA` column in our `df_subject`.

The `Average GPA` column should contain the **weighted average GPA** of each `Subject` by taking into account the **number of students** who received each letter grade in said `Subject`.

In [288…

```
df_subject['Average GPA'] = (4 * (df_subject["A+"]) + 4 * (df_subject["A"]
df_subject
```

Out[288…

Subject	Year	Number	A+	A	A-	B+	B	B-	C+	C
AAS	494134	30940	1050	2829	1053	658	644	308	165	169
ABE	334747	55405	584	3332	912	830	987	260	148	194
ACCY	5737609	1079894	15871	36922	21865	19613	18654	7728	3912	3196
ACE	2290983	320492	8769	19270	6845	5559	9093	3328	2156	3124
ACES	383227	22012	1612	2095	393	197	316	87	92	94
…	…	…	…	…	…	…	…	…	…	…
UP	729992	128783	1483	4977	2508	1437	1509	591	317	339
VB	8040	2489	0	212	6	6	120	1	0	40
VCM	110864	34284	547	1687	35	63	1087	20	23	349
VM	258207	78366	0	2879	8	0	7981	1	0	4725
YDSH	24174	3340	48	105	75	29	36	14	9	7

174 rows × 18 columns

🔬 Test Case Checkpoint 🔬

In [289…

```
## == TEST CASES for Puzzle 2.2 ==
# - This read-only cell contains test cases for your previous cell.
# - If this cell runs without any error or output, you PASSED all test cas
# - If this cell results in any errors, check your previous cell, make cha
import math
assert( len(df_subject) == 174 ), "You shouldn't be changing the length of
assert( "Average GPA" in df_subject.columns), "Make sure your column is st
assert( math.isclose(df_subject['Average GPA'].mean(), 3.47451626135721952

## == SUCCESS MESSAGE ==
# You'll only see this message (With the emoji showing) if you passed all
tada = "\N{PARTY POPPER}"
print(f"{tada} All tests passed! {tada}")
```

```
🎉 All tests passed! 🎉
```

Puzzle 2.3: The "Hardest" and "Easiest" Subjects?

One way we can judge a Subject's difficulty is to consider its **Average GPA**.

Using your `df_subject`'s `Average GPA` column and the two cells below, find:

- The **10** Subjects with the **lowest** `Average GPA` , storing in the DataFrame `hard_subjects`
- The **10** Subjects with the **highest** `Average GPA` , storing the DataFrame `easy_subjects`

In [290…

```
hard_subjects = df_subject.nsmallest(10, "Average GPA")
hard_subjects
```

Out[290…

Subject	Year	Number	A+	A	A-	B+	B	B-	C+	
VM	258207	78366	0	2879	8	0	7981	1	0	4
MATH	5397267	832939	17421	41455	24395	24080	30581	18419	15424	18
TAM	756169	116920	2781	6258	4630	4418	5187	3370	2647	2
SAME	6039	437	2	29	30	20	31	12	14	
PHYS	2068965	296574	18974	20537	13762	12745	13086	11715	9543	9
LAT	10065	510	13	18	15	12	18	6	11	
CHBE	683827	142076	996	4646	2512	2389	4050	1933	1493	2
CHEM	3864516	374605	20644	56056	23207	21953	28416	16803	15106	17
PHIL	1177027	102123	1099	4656	3616	2922	2860	1756	955	
MCB	4862019	614869	10349	18085	14739	13683	15368	9293	6767	6

In [291…

```
easy_subjects = df_subject.nlargest(10, "Average GPA")
easy_subjects
```

Out[291…

Subject	Year	Number	A+	A	A-	B+	B	B-	C+	C	C-	D+	D	
MUSC	93024	21812	250	2551	58	28	55	9	2	14	2	0	5	
CB	4022	1250	0	140	0	0	17	0	0	1	0	0	0	
BUS	869655	62697	1906	10153	1507	652	400	121	58	55	17	17	21	
ERAM	18201	5072	63	138	40	13	9	1	1	0	0	0	1	
NE	2023	100	15	4	6	2	1	0	0	0	0	0	0	
UKR	10070	565	10	206	46	21	9	4	0	1	0	0	0	
CHP	44439	4575	210	299	52	17	24	4	2	6	1	1	0	
REES	6047	603	32	34	7	2	5	2	0	0	0	0	1	
EPOL	216332	50302	506	2821	394	144	99	39	21	23	12	2	15	
EOL	116933	31442	93	1340	264	123	100	12	4	9	0	0	6	

🔬 Test Case Checkpoint 🔬

In [292…

```
## == TEST CASES for Puzzle 2.3 ==
# - This read-only cell contains test cases for your previous cell.
# - If this cell runs without any error or output, you PASSED all test cas
# - If this cell results in any errors, check your previous cell, make cha
import math
assert( len(hard_subjects) == len(easy_subjects) == 10 ), "Make sure you a
assert( math.isclose(hard_subjects['Average GPA'].sum(), 29.85307300716918
assert( math.isclose(easy_subjects['Average GPA'].sum(), 38.5016390326112!

## == SUCCESS MESSAGE ==
# You'll only see this message (With the emoji showing) if you passed all
tada = "\N{PARTY POPPER}"
print(f"{tada} All tests passed! {tada}")
```

```
🎉 All tests passed! 🎉
```

Analysis: "Hardest" and "Easiest" Subjects?

Q3: Observe the `Subject` column of the `hard_subjects` and `easy_subjects` you've found in Puzzle 2.3 above. Do you think these are truly the "Hardest" and "Easiest" subjects at Illinois? Explain why or why not.

Note: You can use http://catalog.illinois.edu/courses-of-instruction/ to find what the different `Subject` codes mean.

Yes, I believe that the subjects listed above in each dataframe for easiest and hardest are actually accurate. This is because some of the hardest courses include PHYS, MATH, CHEM, MCB, which are all really hard subjects with a lot of advanced coursework dealing with advanced math. On the other hand, for the easiest subjects, we have subjects such as MUSC and BUS which are known to be easier majors, definitely easier than PHYS, MATH, etc.

Puzzle 2.4: Visualizing GPA by Subject

We've got the **Average GPA** of each `Subject`, but what if we want to look at the **bigger picture** across all subjects? Well, data visualization comes to the rescue!

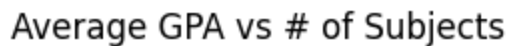
Generate a **histogram** of the **Average GPA** in your `df_subject`. Make sure your histogram looks nice and has the x-axis and y-axis labeled appropriately!

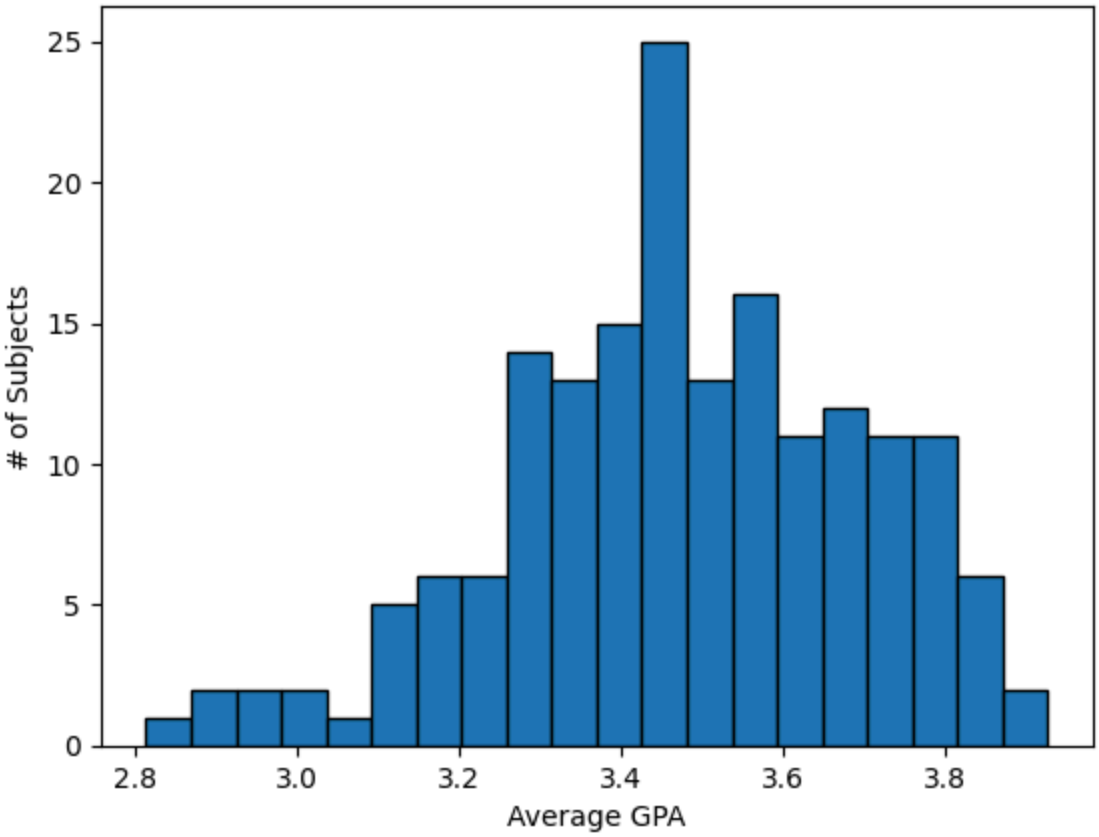
In [293…

```
df_subject["Average GPA"].plot.hist(xlabel = "Average GPA", ylabel = "# o
```

Out[293…

```
<Axes: title={'center': 'Average GPA vs # of Subjects'}, xlabel='Average
GPA', ylabel='# of Subjects'>
```





Analysis: Histogram Interpretations

Q4: Based on the histogram you generated above, what do you think the Average GPA across all courses in a typical Subject at UIUC is? No need for an exact answer, just estimate.

I think the average gpa across all courses in a typical subject at UIUC is around 3.3 because there are some low values that bring the mean down from where most of the data is.

Puzzle 2.5: Your Major!

We've done a lot of analysis on every course and every subject, but Data Science should also be personal to you!

Using your `df_subject`, isolate the row containing the `Subject` of **your Major**, storing in the variable `my_subject`:

(If you're undecided, you can pick any `Subject` you are interested in!)

In [294…

```
my_subject = df_subject.loc["CS"]
my_subject
# the original way df_subject[df_subject["Subject"] == "CS"]
```

Out[294…

```
Year            4.464568e+06
Number          7.273860e+05
```

```
A+              3.340300e+04
A               8.489700e+04
A-              2.913300e+04
B+              2.089500e+04
B               2.297100e+04
B-              1.117900e+04
C+              7.879000e+03
C               9.379000e+03
C-              4.279000e+03
D+              2.338000e+03
D               3.738000e+03
D-              1.343000e+03
F               5.270000e+03
W               1.207000e+03
Students        2.367040e+05
Average GPA     3.381729e+00
Name: CS, dtype: float64
```

Analysis: Your Major's Average GPA

Q5: Observe the `Average GPA` column of your subject from Puzzle 2.5 above. Is it higher or lower than you expected? Explain why!

The average GPA for CS is 3.38 which is higher than I expected, because CS is a hard subject and would have guessed that the average GPA would be more closer to 3 due to the number of advanced and challenging coursework.

Part 3: GPA By Year

At this point, we've investigated the GPA Dataset as a whole and grouped by `Subject`. While our GPA Dataset contains **a lot** of course data, some of the listed courses are quite old: **dating back to 2010**!

One can question the changes to GPA **over time**. Some questions may include:

- Has GPA gone up, because classes became "easier"?
- Has the GPA fallen because of stricter grading policies?
- How was GPA impacted in 2019-2020 at the brunt of COVID?

You will gain some insight into the answers to questions in this section of the lab.

Puzzle 3.1: Exploring Different Years

In the following cell, create a new DataFrame called `df_year` that has a single row for each year. In `df_year`, each **letter grade column** should contain the **total number of students** receiving said grade in that `Year`. Make sure your DataFrame only contains numeric columns.

In [295…
```
df_year = df.groupby("Year").agg("sum", numeric_only = True).reset_index(
```

🔬 Test Case Checkpoint 🔬

In [296…
```
## == TEST CASE for Puzzle 3.1 ==
# - This read-only cell contains test cases for your previous cell.
# - If this cell runs without any errors, you PASSED all test cases!
# - If this cell results in any errors, check your previous cell, make cha
import math
assert( 'df_year' in vars() ), "Make sure your DataFrame grouped by Year
assert( len(df_year) == 15 ), "Make sure you are grouping by 'Year'. There
assert( math.isclose(df_year.Students.mean(), 274749.3333333333) ), "Doub

## == SUCCESS MESSAGE ==
# You will only see this message (with the emoji showing) if you passed a
tada = "\N{PARTY POPPER}"
print(f"{tada} All tests passed! {tada}")
```

🎉 All tests passed! 🎉

Puzzle 3.2: Fixing our `Average GPA` Column Again

Some columns in `df_year` are incorrectly calculated given the way we've grouped the data.

Let's fix this by redefining the `Average GPA` column in our `df_year`.

The `Average GPA` column should contain the **weighted average GPA** of each `Year` by taking into account the **number of students** who received each letter grade in said `Year`.

In [297…
```
df_year['Average GPA'] = (4 * (df_year["A+"]) + 4 * (df_year["A"]) + 3.67
df_year
```

Out[297…

	Year	Number	A+	A	A-	B+	B	B-	C+	C	
0	2010	1493480	19355	77012	36814	30579	43517	17195	11604	15753	5
1	2011	1435567	19569	75526	36462	30987	42453	16776	11351	15844	6
2	2012	822949	11834	44738	20603	17041	22408	9485	6515	8221	3
3	2013	1468576	22541	81114	37513	30624	40241	17471	11611	14875	6
4	2014	1440554	25081	79289	36792	30749	38679	17724	11713	14802	6
5	2015	1516560	29032	87101	39378	31895	40221	18172	12281	14891	6
6	2016	1452803	29094	85548	37962	30278	37678	16759	11590	14199	6
7	2017	1513420	34941	92166	38295	29889	37541	16494	11117	14270	6
8	2018	1509424	40112	98711	39621	29546	35837	16123	10809	13421	5

9	2019	1528969	41642	104139	40875	30890	37332	16680	11203	13623	6
10	2020	1406859	56217	117054	38356	25531	29866	12637	8460	9944	4
11	2021	1615834	67828	125081	42930	29188	32980	14424	9965	12023	5
12	2022	1612502	63636	129419	42224	29724	33629	14653	9778	11950	5
13	2023	1538194	67629	128060	39819	26980	30140	13344	8618	10202	4
14	2024	6462	582	762	197	114	111	48	35	42	

🔬 Test Case Checkpoint 🔬

In [298…

```
## == TEST CASE for Puzzle 3.2 ==
# - This read-only cell contains test cases for your previous cell.
# - If this cell runs without any errors, you PASSED all test cases!
# - If this cell results in any errors, check your previous cell, make cha
import math
assert( len(df_year) == 15 ), "You shouldn't be changing the length of `d
assert( "Average GPA" in df_year.columns), "Make sure your column is stil
assert( math.isclose(df_year['Average GPA'].mean(), 3.3497684856756313) )

## == SUCCESS MESSAGE ==
# You will only see this message (with the emoji showing) if you passed a
tada = "\N{PARTY POPPER}"
print(f"{tada} All tests passed! {tada}")
```

```
🎉 All tests passed! 🎉
```

Puzzle 3.3: Visualizing GPA over Time

We now have the data of the `Average GPA` across the University over time in years. Using this data, we can generate yet another visualization built-in to Pandas to visualize this data: a **line plot**!

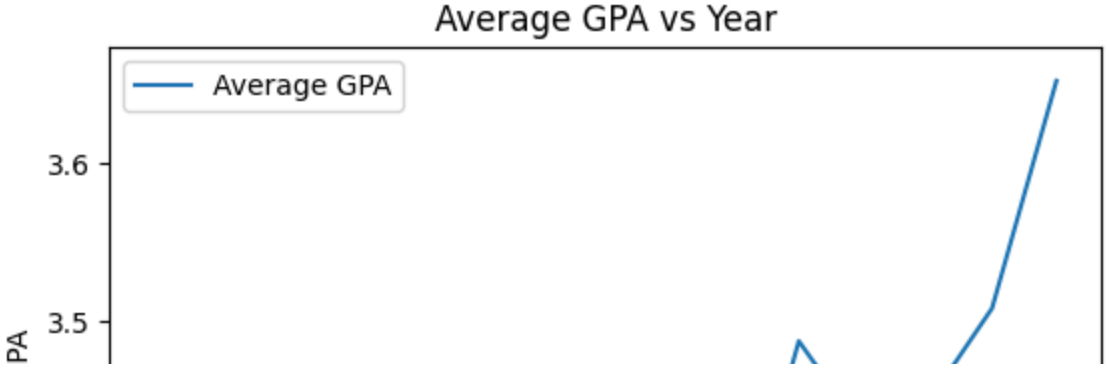
In the cell below, plot the **average GPA over time** using `df_year`. Make sure the x and y axis are labeled appropriately!

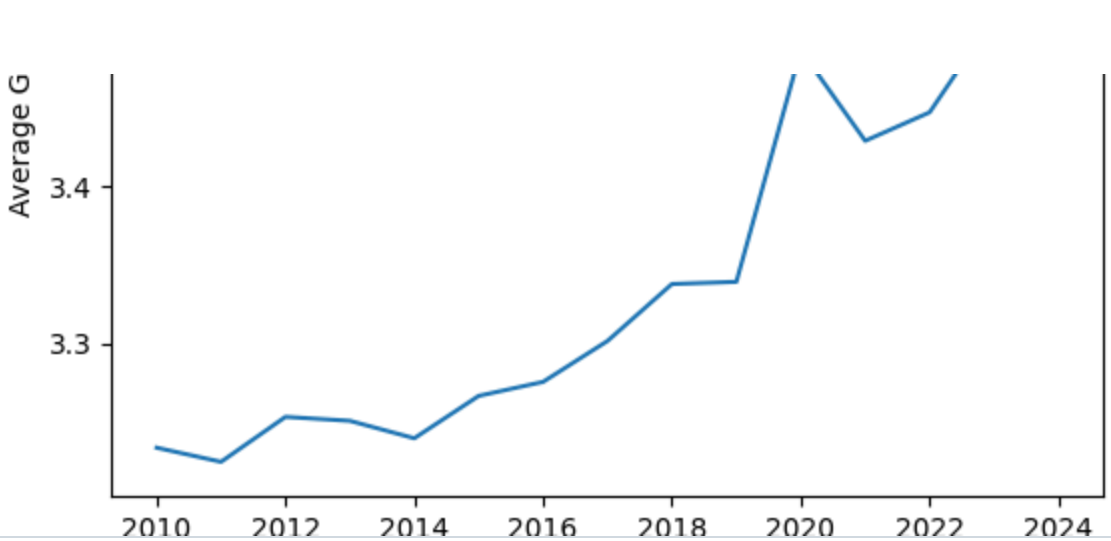
In [299…

```
df_year.plot.line(x="Year", y="Average GPA", title="Average GPA vs Year",
```

Out[299…

```
<Axes: title={'center': 'Average GPA vs Year'}, xlabel='Year', ylabel='Av
erage GPA'>
```




Average G
3.4
3.3
2010
2012
2014
2016
2018
2020
2022
2024