Kaushik Vejju: Student Dataset Analysis & Regression Model

Introduction

The dataset includes student data from a college, including information about a student's id, age, major, and GPA (there are more variables, which are discussed later). The goal of this project is predict the GPA of a student by utilizing different machine learning models. Before evaluating the models, an analysis of the dataset, data preprocessing, and feature engineering steps will be done to extract key features to use in the models.
Predicting GPA is a difficult task since there are many variables outside of the ones included in this dataset that can impact a student's GPA. So, it is important that the right features are extracted from the raw dataset and that any key trends or clusters in the data are appropriately identified and accounted for. In the Data Exploration and Data Cleaning sections, it is imperative that trends in the data and any associations between variables are looked into in order to extract features that will enhance the model's performance.
In general, predicting GPA is a useful action. This can be impactful for university professors or administrators who want to get a sense of grade distribution in classes or understand what factors impact a student's academic performance. With this information, class material and curriculums could possibly be altered to improve student performance.

Data Exploration

To begin, the necessary libraries/packages will be imported. For now, we have imported pandas and matplotlib.pyplot to perform basic data analysis and visualization functions. More libraries, such as ones dealing with hypothesis testing or using models, will be imported later.Using the pd.read_csv() function, the dataset is processed. From here, an exploration of each column of the dataset and possible questions about this dataset will be answered.

In [2]:
```
import pandas as pd
import matplotlib.pyplot as plt # used for providing titles to diagrams
df = pd.read_csv('my_data.csv')
df.head(5)
```

Out[2]:

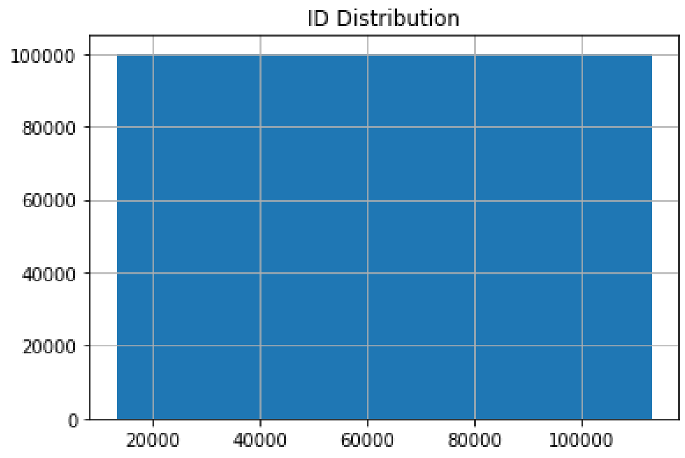
	Unnamed: 0	id	lat	lon	gpa	avg_hours_studied	parents_income	major	tutoring	semester	year	credits	student_age	student_year
0	0	13251	38.878529	-76.776049	3.64	5	68628.72693	Math	No	Fall	2007	15	26	Freshman
1	1	13251	38.878529	-76.776049	3.65	5	68628.72693	Math	No	Spring	2008	15	27	Freshman
2	2	13251	38.878529	-76.776049	4.00	5	68628.72693	Math	No	Fall	2008	9	27	Sophmore
3	3	13251	38.878529	-76.776049	4.00	4	68628.72693	Math	No	Spring	2009	9	28	Sophmore
4	4	13251	38.878529	-76.776049	3.30	4	68628.72693	Math	No	Fall	2009	18	28	Junior

General Analysis On Each Of the Columns In this subsection, information about each column, such as the minimum and maximum values, and a histogram/bar chart will be displayed. The usefulness of the variable is also discussed.

id

In [15]:
```
df['id'].hist()
plt.title("ID Distribution")
```

Out[15]:
```
Text(0.5, 1.0, 'ID Distribution')
```



In [4]:
```
print('The mininum id is', df['id'].min())
print('the max id is', df['id'].max())
```

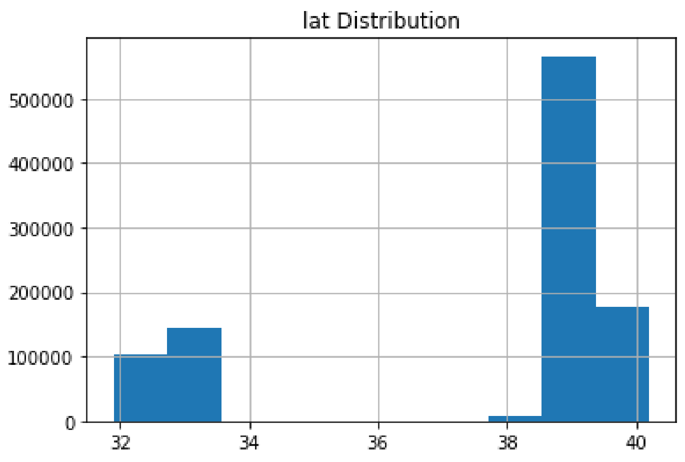
```
The mininum id is 13251
the max id is 113250
```

Is id useful? The id column has a uniform distribution, which makes sense since each student has their own id. Furthermore, multiple rows in the dataset belong to a student with a particular student. This is worth noting when determining how long a student attended this college or how their GPA or credits have changed throughout their time in college. Although it wouldn't make sense to find the 'mean' ID and look into summary stats, the id is helpful when identifying students and finding metrics about a particular student. For this reason, the id column isn't very useful for hypothesis testing or training models, but it does reveal some insights about the data, as mentioned above.

lat & lon

In [17]:
```
df['lat'].hist()
plt.title("lat Distribution")
```

Out[17]:
```
Text(0.5, 1.0, 'lat Distribution')
```

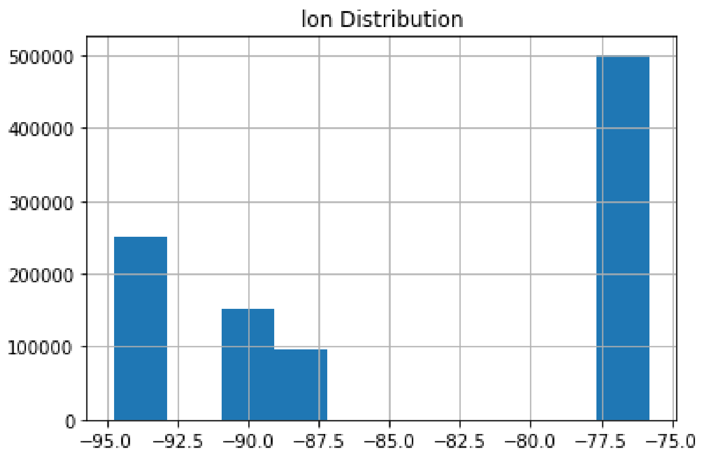


In [18]:

```
df['lon'].hist()
plt.title("lon Distribution")
```

Out[18]:

```
Text(0.5, 1.0, 'lon Distribution')
```



In [7]:

```
print('The mininum longitude is', df['lon'].min())
print('the max longitude is', df['lon'].max())
print('The mininum latitude is', df['lat'].min())
print('the max latitude is', df['lat'].max())
```

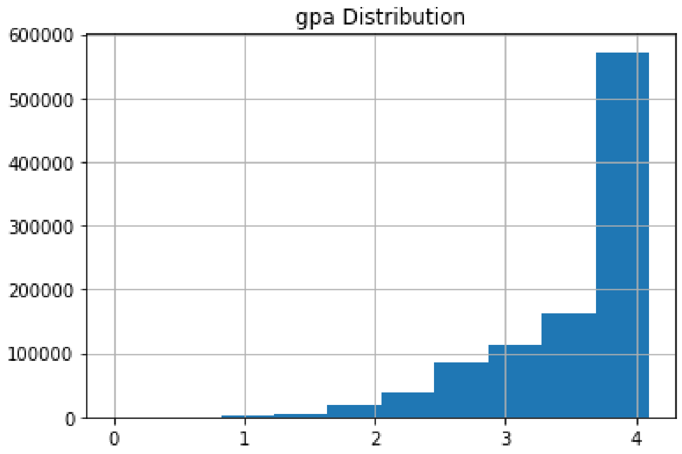
```
The mininum longitude is -94.75403348152076
the max longitude is -75.79067949290653
The mininum latitude is 31.89954590722222
the max latitude is 40.19658486280609
```

Are lat & lon useful? The latitude and longitude are interesting variables to consider. Looking at the histograms above, there are clear clusters in the latitude and longitude. These clusters can reveal interesting insights about the location of this college and where its students are from. These can be features that can be provided to the models. So, latitude and longitude are useful variables to consider.

gpa

In [19]:
```
df['gpa'].hist()
plt.title("gpa Distribution")
```

Out[19]:
```
Text(0.5, 1.0, 'gpa Distribution')
```



In [9]:
```
print('The mininum gpa is', df['gpa'].min())
print('the max gpa is', df['gpa'].max())
```

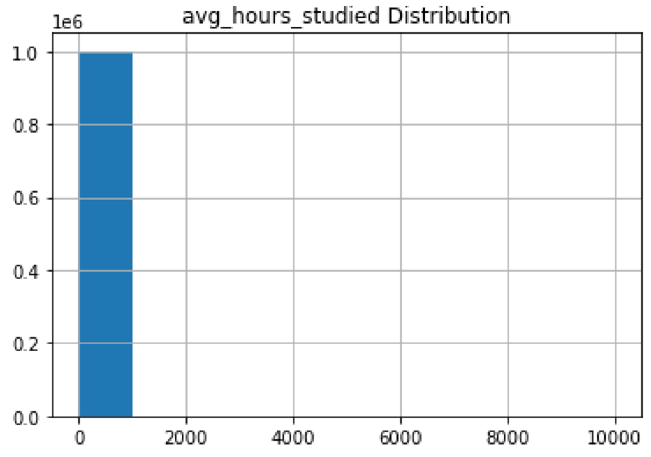
```
The mininum gpa is 0.0
the max gpa is 4.1
```

is GPA useful?: GPA is a useful variable since the goal of the model to predict the GPA. So, it is important to find any relations between the gpa and other variables, such as the income of a students parent, the amount of time students studied, etc. The gpa can also be used for hypothesis testing, such as determining if grade inflation exists or if different majors have different gpa distributions. From the histogram above, its worth noting many students have high GPAs. Many entries in this dataset have GPAs that exceed 3.0, which is an aspect of the dataset to consider.

avg_hours_studied

In [21]:
```
df['avg_hours_studied'].hist()
plt.title("avg_hours_studied Distribution")
```

Out[21]:
```
Text(0.5, 1.0, 'avg_hours_studied Distribution')
```



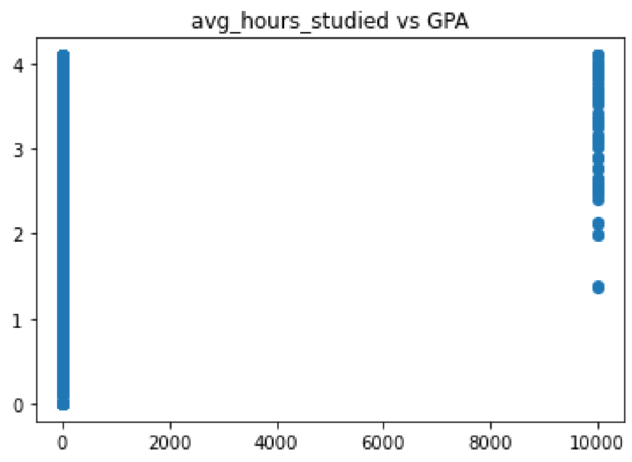
In [11]:
```
print('The mininum avg_hours_studied is', df['avg_hours_studied'].min())
print('the max avg_hours_studied is', df['avg_hours_studied'].max())
```

```
The mininum avg_hours_studied is 0
the max avg_hours_studied is 10000
```

is avg_hours_studied useful? avg_hours_studied is an interesting variable as it extreme data points that are not possible. In other words, it is not possible for someone to study 10,000 hours in a week. So, this column will need to be modified in the data cleaning section. Regardless, avg_hours_studied is a useful metric since it could possible tie in with the GPA, which is what the model is trying to predict. For example, a greater avg_hours_studied could mean a higher GPA. Let's plot this relationship below:

In [4]:
```
plt.scatter(x=df['avg_hours_studied'], y=df['gpa'])
plt.title("avg_hours_studied vs GPA")
```

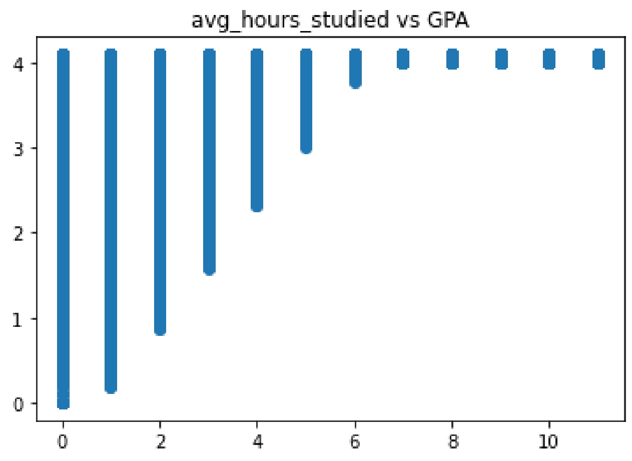
Out[4]:
```
Text(0.5, 1.0, 'avg_hours_studied vs GPA')
```



Due to the outlier of 10,000 hours the relationship between the avg_hours_studied and GPA is difficult to see. For this reason, the scatter plot will be remade below, this time with the 10,000 filtered out.

```
In [5]:
copy_df = df[df['avg_hours_studied']<10000]
plt.scatter(x=copy_df['avg_hours_studied'], y=copy_df['gpa'])
plt.title("avg_hours_studied vs GPA")
```

```
Out[5]: Text(0.5, 1.0, 'avg_hours_studied vs GPA')
```



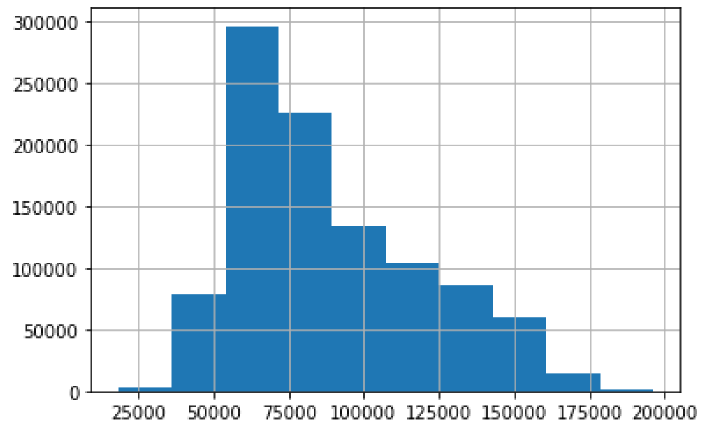
From this scatter plot, it is observed that as the avg_hours_studied increases, the GPA increases as well. This relationship is important to consider moving forward, particulary when choosing features for the models to work with.

parents_income

```
In [47]:
df['parents_income'].hist()
```

Out[47]:

```
<AxesSubplot:>
```



In [48]:

```
print('The mininum parents_income is', df['parents_income'].min())
print('the max parents_income is', df['parents_income'].max())
```

```
The mininum parents_income is 18229.30176674215
the max parents_income is 196273.33724594
```

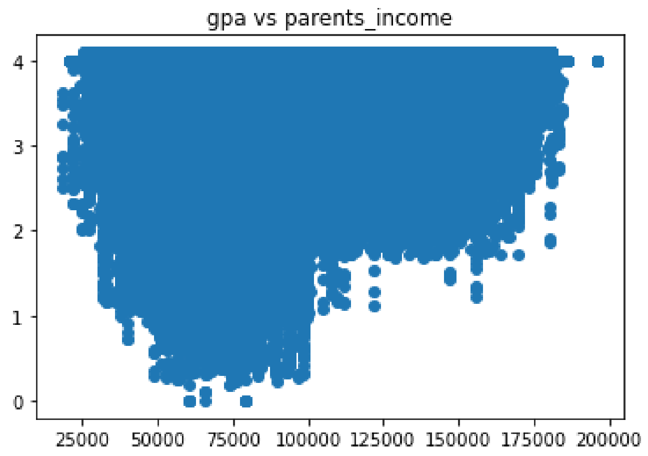
is parents_income useful?: parents_income is also a useful variable. This distribution has a positive skew and the range of parents_income means that this variable can be normalized during the data cleaning process. Because it is numerical, parents_income would be useful for models like KNN, which perform well with numerical information. parents_income could also have a potential relation to the gpa, where a higher parents_income could lead to a higher gpa. This relationship is displayed below with a scatterplot.

In [26]:

```
plt.scatter(x=df['parents_income'], y=df['gpa'])
plt.title("gpa vs parents_income")
```

Out[26]:

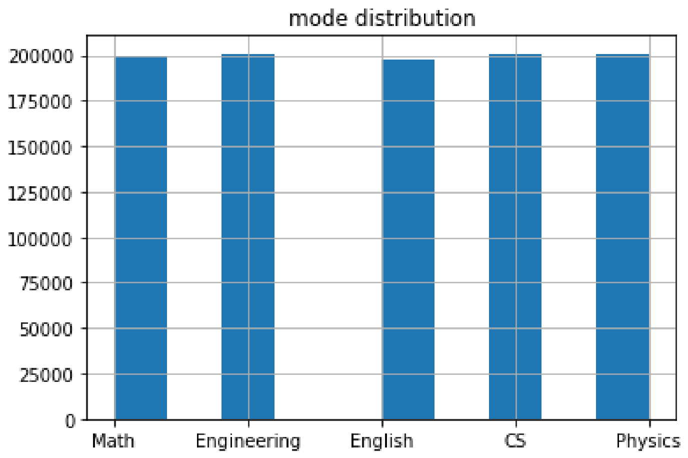
```
Text(0.5, 1.0, 'gpa vs parents_income')
```



major

In [27]:
```
df['major'].hist()
plt.title("mode distribution")
```

Out[27]:
```
Text(0.5, 1.0, 'mode distribution')
```



In [50]:
```
df.groupby('major').size()
```

Out[50]:
```
major
CS             200505
Engineering    200787
English        198125
Math           199750
Physics        200833
dtype: int64
```

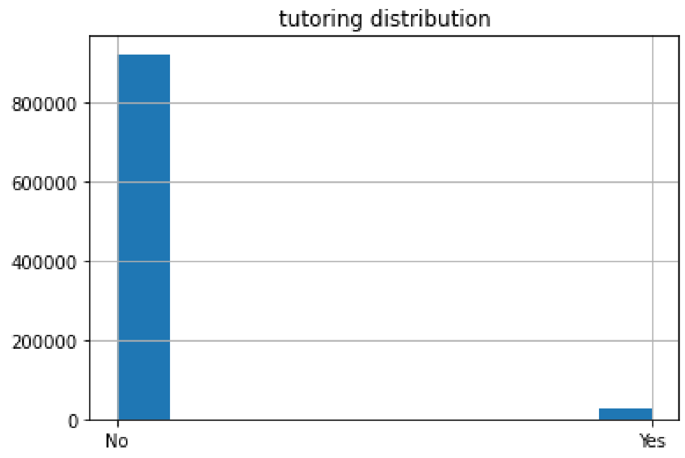
Most students are Engineering majors while the the least common major is English.

is major useful? interestingly, the amount of people in each major of this dataset doesn't differ significantly between majors. major could be a useful variable as well since the type of major a student is in could influence their GPA. major also is used in hypothesis tests, such as determining if different majors have different gpa distributions. It can also be used to predict the GPA of a student (by knowing their major). So, major is a useful variable.

tutoring

In [28]:
```
df['tutoring'].hist()
plt.title("tutoring distribution")
```

Out[28]:
```
Text(0.5, 1.0, 'tutoring distribution')
```



In [52]:
```
df.groupby('tutoring').size()
```

Out[52]:
```
tutoring
No     923106
Yes     27396
dtype: int64
```

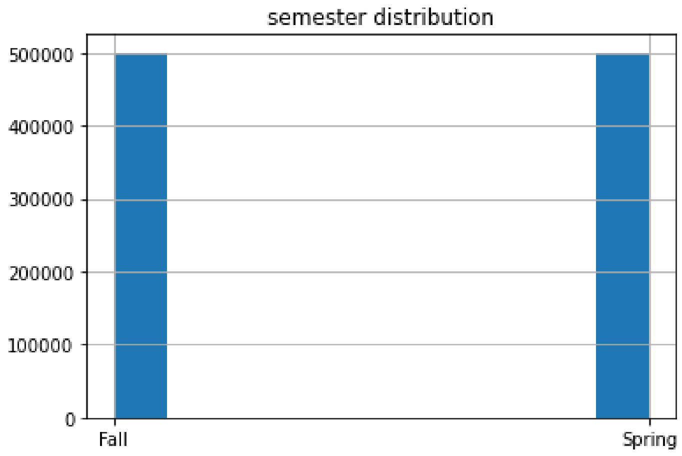
Most students are not enrolled in tutoring while some are.

is tutoring useful? Since most students are not tutored, tutoring may not be a very useful variable. However, it can be worth seeing if tutoring improves a student's GPA, or if there is difference in the GPAs between students who have been tutored vs those who have not. Based on the results of the hypothesis test, tutoring can be used as a feature for the models. More analysis on this variable will be done to see if it will be useful for our model for predicting the GPA of a student.

semester

In [29]:
```
df['semester'].hist()
plt.title("semester distribution")
```

Out[29]:
```
Text(0.5, 1.0, 'semester distribution')
```



In [54]:
```
df.groupby('semester').size()
```

Out[54]:
```
semester
Fall      500000
Spring    500000
dtype: int64
```

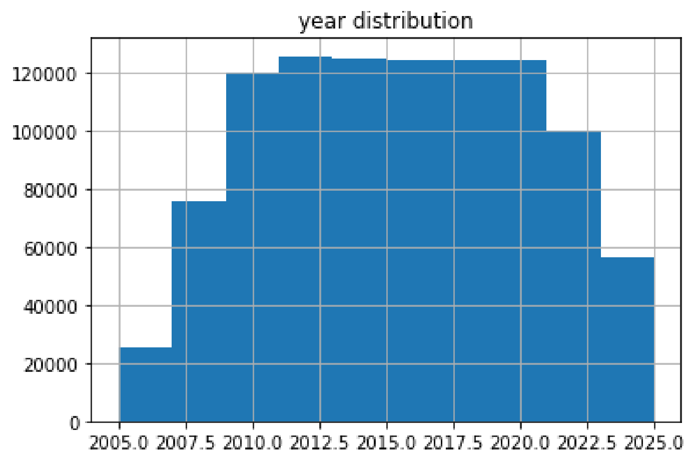
No difference between the amount of students enrolled in the Fall vs those enrolled in the Spring.

Is semester useful? semester may not be a useful variable. While it does give an indication of time, using an actual year is better instead for a time analysis. Furthermore, it is hard to draw any relation between the semester and other variables. For instance, recognizing a relation between the semester and gpa could be difficult and may not even be needed. Further exploration of the data is needed to determine if semester is a variable worth using to predict GPA.

year

In [31]:
```
df['year'].hist()
plt.title("year distribution")
```

Out[31]:
```
Text(0.5, 1.0, 'year distribution')
```



In [56]:

```
df.groupby('year').size()
```

Out[56]:

```
year
2005     6392
2006    19019
2007    31512
2008    44111
2009    56834
2010    63014
2011    62820
2012    62763
2013    62505
2014    62263
2015    62120
2016    61997
2017    61936
2018    62002
2019    62107
2020    62246
2021    56164
2022    43789
2023    31382
2024    18796
2025     6228
dtype: int64
```

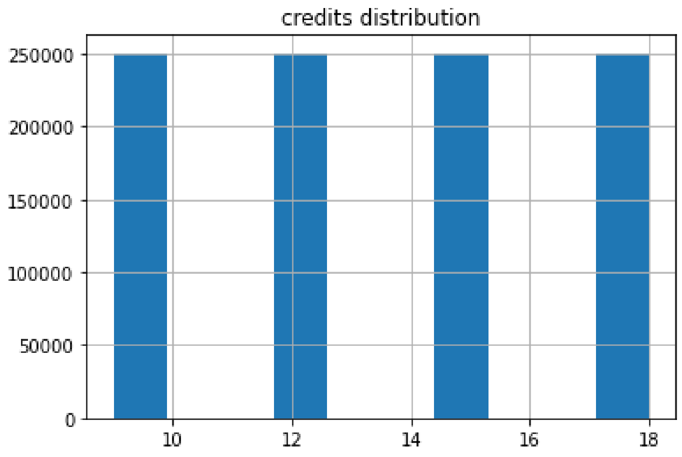
The years range from 2005 (min) to 2025 (max). Year with highest number of entries is 2010 and the year with the lowest number of entries is 2025.

Is year useful? the year is partially useful variable. It can be used for any time series analysis, such as how GPA changes over time (helpful for checking for GPA inflation). Other than that, the year is not entirely useful. If there is GPA inflation, perhaps the year can help predict GPA, but it wouldn't be the only feature used to make this prediction.

credits

In [32]:
```
df['credits'].hist()
plt.title("credits distribution")
```

Out[32]:
```
Text(0.5, 1.0, 'credits distribution')
```



In [58]:
```
print('The mininum credits is', df['credits'].min())
print('the max credits is', df['credits'].max())
```

```
The mininum credits is 9
the max credits is 18
```

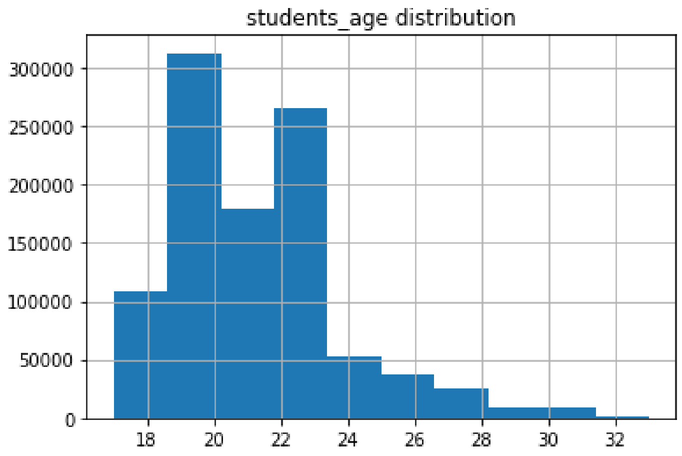
is credits useful? credits may not be a very useful variable. Compared to other variables like the avg_hours_studied, it may not be related to the GPA. The number of credits a student takes could influence their GPA, but there are better features that can be used to predict the GPA. credits also doesn't also give an indication of time, making it less useful compared to other variables.

- Just a side note, but it also appears that the credits follow multiples of 3: 9, 12, 15, and 18. This can help address one of the questions, which is to see if this college offers any one credit courses.

student_age

In [33]:
```
df['student_age'].hist()
plt.title("students_age distribution")
```

Out[33]:
```
Text(0.5, 1.0, 'students_age distribution')
```



In [60]:
```
print('The mininum age is', df['student_age'].min())
print('the max age is', df['student_age'].max())
```

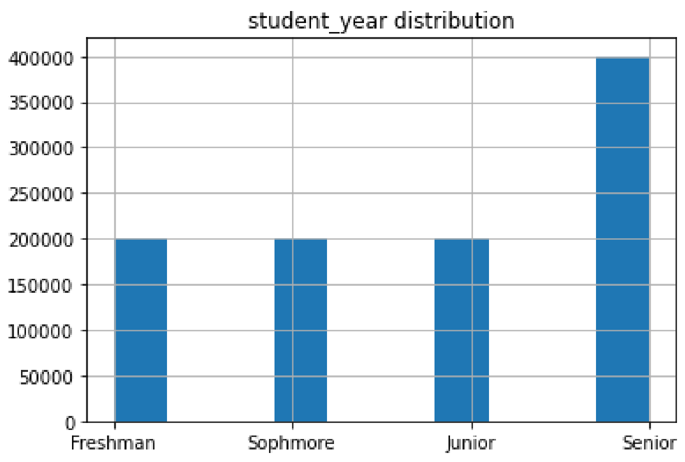
```
The mininum age is 17
the max age is 33
```

Is student_age useful? student_age could be a useful variable. While the age could influence one's GPA, a variety of other factors, like a students major, or hours studied could be better indicators of their GPA. Age can be useful for examining the student body or how many credits a student could take (older students may have jobs and be taking less credits), but this is not our main concern for this dataset.

student_year

In [34]:
```
df['student_year'].hist()
plt.title("student_year distribution")
```

Out[34]:
```
Text(0.5, 1.0, 'student_year distribution')
```



In [35]:
```
df.groupby('student_year').size()
```

Out[35]:
```
student_year
Freshman    200000
Junior      200000
Senior      400000
Sophmore    200000
dtype: int64
```

Most entries are in the senior category (max), while the rest (min) are evenly distributed amongst Freshman, sophomore, and junior.

is student_year useful? student_year may be partially useful. For instance, different classes could have different GPA distributions, which can be helpful for predicting one's GPA. However, other features that were discussed earlier can be better predictors since they are numerical and are more closely related to GPA outcome. However, the class standing may give an indication of transfer students and how long a student has remained in university.

Possible Questions For This Dataset

Does the school have transfer students?

Before determining if the school has tranfer students, a definition of a transfer student is provided:
A transfer student is someone who takes credits at another institution and transfers into another university. Often times, they will have a higher class standing since they have enrolled in more credits.
To determine if there are transfer students, take each student (which is identified by their ID) and see their class standing when they start at this school. If their class standing is not Freshman, then they could be a transfer student. The code for this is shown below:

**** Note that multiple entries in the dataset can correspond to a student. So, the first entry for a student wih a unique student ID represents the first semester they started in this university.**

In [36]:
```
grouped_by_student = df.groupby("id")
'''
```

```
Group the entries above using groupby(). With the first() function shown below, the first row
for each student (or id) is obtained. This is useful because it contains the information for when the student first started at this
college. The main focus is the student_year (freshman, sophmore, etc), which will be shown in the next cell.
'''
student_starts = grouped_by_student.first()
student_starts.head(5)
```

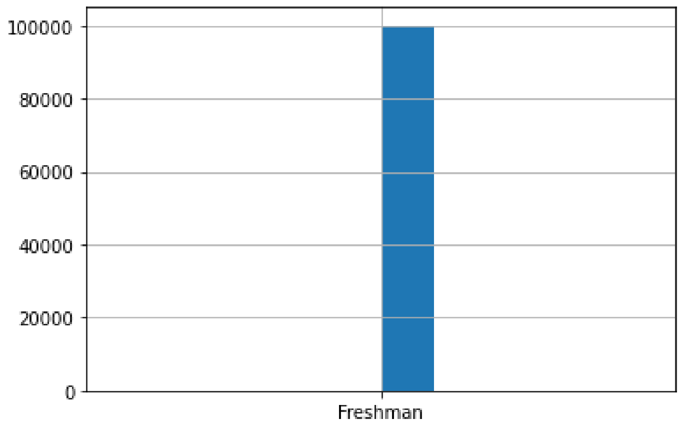
Out[36]:

	Unnamed: 0	lat	lon	gpa	avg_hours_studied	parents_income	major	tutoring	semester	year	credits	student_age	student_year
id													
13251	0	38.878529	-76.776049	3.64	5	68628.726930	Math	No	Fall	2007	15	26	Freshman
13252	10	39.393276	-93.623221	3.62	5	57884.268240	Engineering	No	Fall	2007	12	19	Freshman
13253	20	39.313355	-93.858182	3.70	5	69090.108548	Math	No	Fall	2006	18	22	Freshman
13254	30	38.955780	-76.506825	3.04	4	89732.378369	English	No	Fall	2013	18	18	Freshman
13255	40	39.125074	-93.960881	3.88	5	108918.081738	Math	No	Fall	2017	15	19	Freshman

From the data above, plot a histogram that shows the student_year of when each student began at this college.

In [64]:
```
student_starts['student_year'].hist()
```

Out[64]:
```
<AxesSubplot:>
```



As displayed by the histogram above, all of the students started this college as freshmen. It can be concluded that there are no transfer students in this college.

What is the median length of attendence at this university?

Here is an interesting finding:

In [65]:
```
df.head(11)
```

Out[65]:

	Unnamed: 0	id	lat	lon	gpa	avg_hours_studied	parents_income	major	tutoring	semester	year	credits	student_age	student_year
0	0	13251	38.878529	-76.776049	3.64	5	68628.72693	Math	No	Fall	2007	15	26	Freshman
1	1	13251	38.878529	-76.776049	3.65	5	68628.72693	Math	No	Spring	2008	15	27	Freshman
2	2	13251	38.878529	-76.776049	4.00	5	68628.72693	Math	No	Fall	2008	9	27	Sophmore
3	3	13251	38.878529	-76.776049	4.00	4	68628.72693	Math	No	Spring	2009	9	28	Sophmore
4	4	13251	38.878529	-76.776049	3.30	4	68628.72693	Math	No	Fall	2009	18	28	Junior
5	5	13251	38.878529	-76.776049	4.00	4	68628.72693	Math	No	Spring	2010	9	29	Junior
6	6	13251	38.878529	-76.776049	3.41	4	68628.72693	Math	No	Fall	2010	18	29	Senior
7	7	13251	38.878529	-76.776049	3.96	4	68628.72693	Math	No	Spring	2011	12	30	Senior
8	8	13251	38.878529	-76.776049	3.40	4	68628.72693	Math	No	Fall	2011	18	30	Senior
9	9	13251	38.878529	-76.776049	3.99	3	68628.72693	Math	No	Spring	2012	12	31	Senior
10	10	13252	39.393276	-93.623221	3.62	5	57884.26824	Engineering	No	Fall	2007	12	19	Freshman

Notice how above, there is a student with the ID 13251, who starts college as a Freshman in the Fall of 2007 and continues all the way till Spring 2012. The rows in the dataset are organized in manner where the next row represents the next semester for a student with a particular id. By finding the number of rows for each student ID, the average length (in semesters) of attendance for each student can be determined. This is shown by the code below:

In [3]:

```python
# Group by the id, and find the number of rows (representing the number of semesters) by using the .size() function
id_years = df.groupby('id').size()
id_years
```

Out[3]:

```
id
13251     10
13252     10
13253     10
13254     10
13255     10
          ..
113246    10
113247    10
113248    10
113249    10
113250    10
Length: 100000, dtype: int64
```

In [4]:

```python
# Find the median by simply appying the median() function to the id_years variable
id_years.median()
```

Out[4]:

```
10.0
```

Do you think this university has any one credit classes?

Earlier, it was stated that the university has courses with multiples of 3 (9, 12, 15, 18). It can be assumed that the number of credits for each class at this college is 3. Under this assumption, for each entry in the dataset, a modulus operator will be applied to see if the number of credits a student takes is divisible by 3. If so, than they are taking no 3 credit classes. If not, then the result of modulus operation will be 1, which indicates the prescence of a 1 credit class.

To check for the condition above, a mask on this dataset is applied:

In [68]:
```
one_credit_courses = df[df['credits'] %  3 == 1]
one_credit_courses
```

Out[68]:

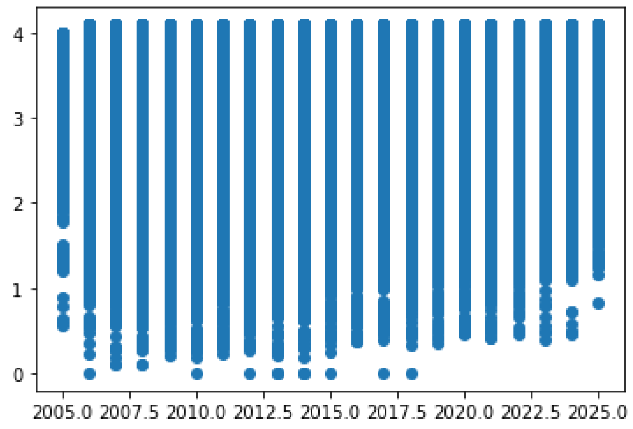
Unnamed: 0	id	lat	lon	gpa	avg_hours_studied	parents_income	major	tutoring	semester	year	credits	student_age	student_year

After applying mask, 0 rows from the original dataset are displayed. **So, it can be concluded that the college does not have any one credit classes.**

Is grade inflation a problem at this university?

To check for grade inflation, a scatter plot of year vs GPA can be made to observe how GPA has changed over time. **In general, grade inflation is defined as scenario in which there is an increase in the average GPA for a student body over time.** This can be examined first with a scatter plot, but a hypothesis test will also be performed to make more valid conclusions.

In [40]:
```
import matplotlib.pyplot as plt
plt.scatter(x=df['year'], y=df['gpa'])
plt.show()
```



This scatter plot does reveal something interesting. It appears that in later years, (2022-2025), the mininum GPA's of each year are increasing (which means that GPAs are getting better). However, this is not enough to conclude grade inflation. **One way to check for grade inflation is to seperate the data into two groups:**

- One group will be the entries from 2005-2015,

- Another group will be entries from 2016-2025.

By comparing the average GPA in these groups through a T-test, a conclusion regarding grade inflation can be made.

First, to actually run this test, another column, called 'year_category' will be added to categorize the years.

In [70]:
```
df['year_category'] = df['year'].apply(lambda x: '2005-2015' if x <= 2015 else '2016-2025')
df.head()
```

Out[70]:

	Unnamed: 0	id	lat	lon	gpa	avg_hours_studied	parents_income	major	tutoring	semester	year	credits	student_age	student_year	year_catego
0	0	13251	38.878529	-76.776049	3.64	5	68628.72693	Math	No	Fall	2007	15	26	Freshman	2005-20
1	1	13251	38.878529	-76.776049	3.65	5	68628.72693	Math	No	Spring	2008	15	27	Freshman	2005-20
2	2	13251	38.878529	-76.776049	4.00	5	68628.72693	Math	No	Fall	2008	9	27	Sophmore	2005-20
3	3	13251	38.878529	-76.776049	4.00	4	68628.72693	Math	No	Spring	2009	9	28	Sophmore	2005-20
4	4	13251	38.878529	-76.776049	3.30	4	68628.72693	Math	No	Fall	2009	18	28	Junior	2005-20

Now, let's apply a T-test, where the average GPA of these two groups will be comapared. Shown below are the hypotheses:

- H0: The average GPA in the 2005-2015 category is the same as the average GPA in the 2016-2025 category.
- HA: The average GPA in the 2005-2015 category < the average GPA in the 2016-2025 category (indicating grade inflation)

In [71]:
```
from scipy.stats import ttest_ind
group1 = df[df['year_category']=='2005-2015']
group2 = df[df['year_category']=='2016-2025']

ttest_ind(group1['gpa'], group2['gpa'], equal_var = False)
```

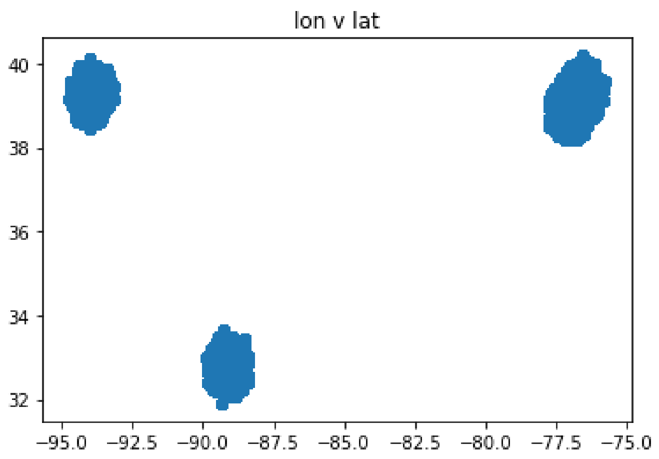
Out[71]:
```
Ttest_indResult(statistic=-73.72532182696443, pvalue=0.0)
```

From this t-test, a p-value of 0.0 is obtained. This is a very small p-value which means that the null hypothesis can be rejected. There is enough statistically significant evidence to conclude that the the average GPA in the 2005-2015 category is less than the average GPA in the 2016-2025 category. So, it can be concluded that grade inflation is present.

In what area is this university located?

Using a scatterplot, the distribution of the lon and lat points can be observed:

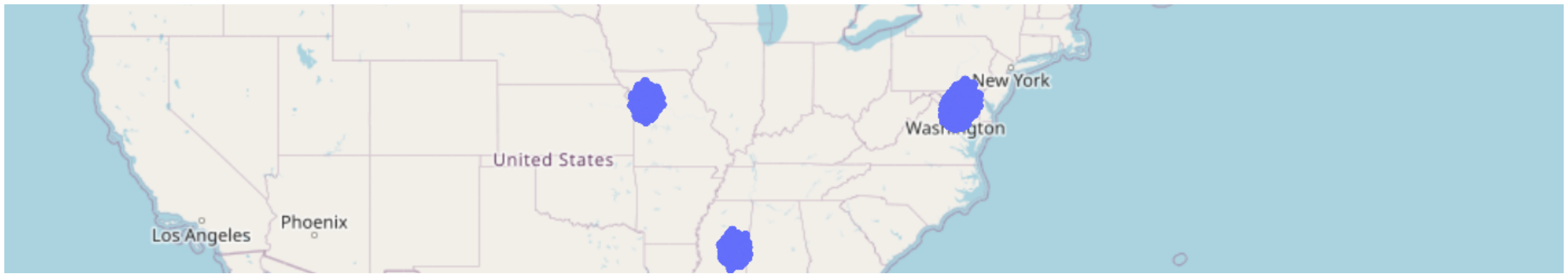
In [42]:
```
plt.scatter(x=df['lon'], y=df['lat'])
plt.title("lon v lat")
plt.show()
```



There are three clusters, as shown above. A map, provided by the plotly.express library, can be utilized to see the geographical location of these clusters.

In [5]:
```
import plotly.express as px

fig = px.scatter_mapbox(df, lat="lat", lon="lon", zoom=3, height=300)
fig.update_layout(mapbox_style="open-street-map")
fig.show()
```



Based on the plot above, it can be seen that there are three clusters of students. One cluster is in Missouri, another in Maryland area, and another in Mississipi. Let us check to see the number of points in each cluster. By observing the plots above, it can be seen that the clusters clearly differ in their longtitudes. By dividing clusters by this metric, the number of entries belonging to a cluster is displayed.

```
In [73]:
cluster_1 = df[df['lon'] < -91]
len(cluster_1)
```

```
Out[73]: 250390
```

```
In [74]:
cluster_2 = df[(df['lon'] > -91) & (df['lon'] < -85)]
len(cluster_2)
```

```
Out[74]: 248720
```

```
In [75]:
cluster_3 = df[df['lon'] > -85]
len(cluster_3)
```

```
Out[75]: 500890
```

The information above reveals that a majority of the students are in cluster 3, which is near the Maryland area.
Because of these 3 distinct clusters, it can be stated that this college could have 3 distinct campuses in the United States. The main campus is in the Maryland area, as shown by the higher number of students compared to the other regions. The other two campuses are located in Missouri and Mississipi, but they have a smaller number of students.

How often do students switch majors?

To do this, group the rows in the dataset by id, to get the individual students. Next, apply the unique() function on each of the students majors to see the majors that each student has enrolled in.

```
In [45]:
res = df.groupby('id')['major'].unique()
res
```

```
Out[45]:
id
13251              [Math]
13252       [Engineering]
13253              [Math]
13254           [English]
13255              [Math]
                  ...
113246          [Physics]
113247             [Math]
113248      [Engineering]
113249         [Math, CS]
113250          [English]
Name: major, Length: 100000, dtype: object
```

Now that this operation is done, loop through the series above and find the number of arrays that have more than one element. This indicates that the students has switched their major. Take this number and divide it by 100,000 which is the total number of students in this dataset.

In [46]:

```
switch = 0; # keeps track of a student who has switched their major once
for x in res:
    if len(x) > 1:
        switch += 1
switch/100000
```

Out[46]:

```
0.06785
```

6. Does tutoring make a statistically significant difference in GPA?

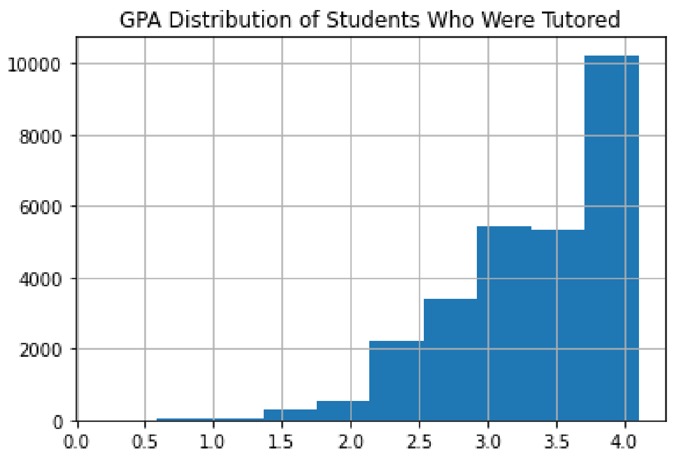
First, divide the entries in the dataset based on if they are doing tutoring or not. Then, compare the GPA distributions of the different values of the 'tutoring' category.

In [47]:

```
tutoring_df = df[df['tutoring']=='Yes']
tutoring_df['gpa'].hist()
plt.title("GPA Distribution of Students Who Were Tutored")
```

Out[47]:

```
Text(0.5, 1.0, 'GPA Distribution of Students Who Were Tutored')
```

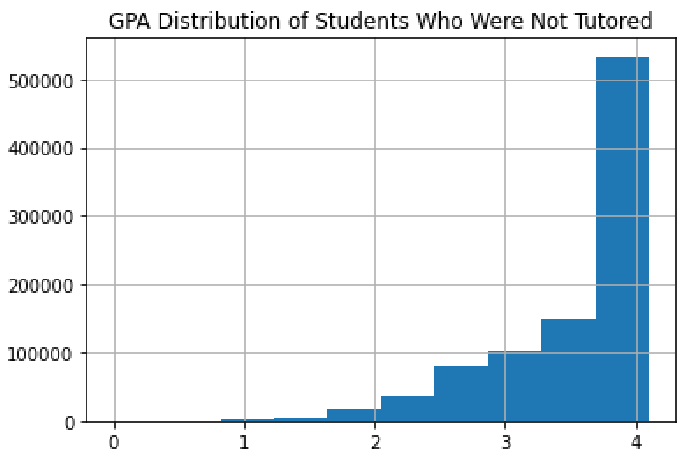


In [48]:

```
no_df = df[df['tutoring']=='No']
no_df['gpa'].hist()
plt.title("GPA Distribution of Students Who Were Not Tutored")
```

Out[48]:

```
Text(0.5, 1.0, 'GPA Distribution of Students Who Were Not Tutored')
```



T-test

For this problem, a T-Test is conducted. This is because the average GPA of two distinct groups: entries where a student used tutoring and entries where a student has not been tutored, is being compared.

From this, we can set up our hypothesis:

- H0: the average GPA of students who had tutoring is the same as the average GPA of students who did not have tutoring
- HA: the average GPA of students who had tutoring != average GPA of students who did not have tutoring

From the scipy library, import the ttest:

In [50]:
```
from scipy.stats import ttest_ind
```

In [51]:
```
group1 = df[df['tutoring']=='Yes']
group2 = df[df['tutoring']=='No']

ttest_ind(group1['gpa'], group2['gpa'])
```

Out[51]:
```
Ttest_indResult(statistic=-54.84689203285616, pvalue=0.0)
```

From the t-test above, a p-value of 0.0 is obtained. This is a very low p-value, which means that the null hypothesis is rejected. **There is statistically significant evidence that the average GPA of students who received tutoring is different than that of students who did not recieve tutoring.**

Data Cleaning

First, missing values in the dataset will be dealt with. By addressing any missing data, potential bias can be reduced and the effectiveness/validity of the models is enhanced. To start, the columns/variables in this dataset with missing entries will be determined first.

In [7]:
```
# List out the columns:
df.columns
```

Out[7]:
```
Index(['Unnamed: 0', 'id', 'lat', 'lon', 'gpa', 'avg_hours_studied',
       'parents_income', 'major', 'tutoring', 'semester', 'year', 'credits',
       'student_age', 'student_year'],
      dtype='object')
```

In [8]:
```
# With this loop, an array containing columns with missing entries will be outputed:
null_arr = []
for c in df.columns:
    if df[c].isnull().any():
        null_arr.append(c)
null_arr
```

Out[8]:
```
['tutoring']
```

By applying the loop above, it appears that 'tutoring' is the only column that has NaN values. This detail is displayed below:

In [9]:
```
missing_tutoring_rows = df[df['tutoring'].isnull()]
missing_tutoring_rows.head(5)
```

Out[9]:

	Unnamed: 0	id	lat	lon	gpa	avg_hours_studied	parents_income	major	tutoring	semester	year	credits	student_age	student_year
25	25	13253	39.313355	-93.858182	3.67	4	69090.108548	Math	NaN	Spring	2009	15	25	Junior
71	71	13258	39.156290	-77.105080	4.00	6	154662.439742	Physics	NaN	Spring	2008	12	20	Freshman
73	73	13258	39.156290	-77.105080	4.00	6	154662.439742	Physics	NaN	Spring	2009	12	21	Sophmore
79	79	13258	39.156290	-77.105080	4.00	4	154662.439742	Physics	NaN	Spring	2012	12	24	Senior
192	192	13270	39.538076	-76.738201	4.00	6	100241.332215	Engineering	NaN	Fall	2013	9	18	Sophmore

One way to deal with these NaN values is to replace them with a value like "Not Provided". This can give another category to work with for the tutoring colum rather than deleting the rows from the dataset. In addition, the original sample size and key information from other columns, such as the students age and credits, is kept. This also removes any form of bias that comes with removing missing values.</b>

In [10]:
```
df['tutoring'] = df['tutoring'].fillna("Not Provided")
```

Missing values have been dealt with. **Now, outliers need to be handled effectively.**

In the previous section, histograms for the different variables were plotted. One notable histogram was average_hours_studied, which contained an interesting outlier. By applying the groupby() function for the average_hours_studied column, this outlier can be examined in greater detail.

In [11]:
```
df.groupby('avg_hours_studied').size()
```

Out[11]:
```
avg_hours_studied
0          10111
1          29633
2          81280
3         169294
4         244714
5         219804
6         131354
7          70190
8          33043
9           9002
10          1411
11            62
10000        102
dtype: int64
```

After grouping the avg_hours_studied column, it can be seen that there are 102 entries where the average hours studied is 10,000. Since there are only 168 hours in a week, such a number is not possible.

Since there are 102 entries (out of 1 million) with 10,000 hours, **removing these entries is one option** to consider since they are a very small portion of the data. In addition, entering 10,000 can be a data entry error, so it is not worth it to work with these outliers and use them for the model to make GPA predictions.

The code to remove these outliers is shown below:

In [12]:
```
# make sure that the avg_hours_studied each week is <= 168
df = df[df['avg_hours_studied'] <= 168]
```

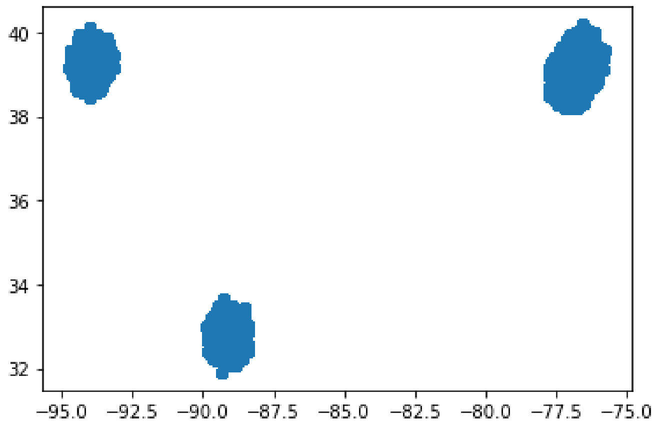
Now, one-hot-encoding for any categorical variables will be done by using the pd.get_dummies() function. This is a useful step as it allows the models to work with categorical information that be helpful to make predictions. In addition, one-hot encoding removes any form of rank associated with a categories values.

The categorical variables are major, tutoring, semester, and student_year:

In [14]:
```
# One-hot encoding for the categorical variables
df[['CS', 'Engineering', 'English', 'Math', 'Physics']]=pd.get_dummies(df['major'])
df[['No', 'Not Provided', 'Yes']]=pd.get_dummies(df['tutoring'])
df[['Fall', 'Spring']] = pd.get_dummies(df['semester'])
df[['Freshman', 'Junior', 'Senior', 'Sophmore']] = pd.get_dummies(df['student_year'])
```

Latitude and longtitude need to be handled as well. One earlier finding was that 3 clusters form when the latitude and longtitude points are plotted. These clusters are seperated by their longitude, as shown below:

In [15]:
```
import matplotlib.pyplot as plt
plt.scatter(x=df['lon'], y=df['lat'])
plt.show()
```



A new column, called "Region", will be created. The region is determined by the longitude value. Here is how the regions can be split:

- long < -91: Missouri
- long > -91 & long < -85: Mississipi
- long > -85: Maryland

The code to do this is shown below:

```
In [16]: df['Region'] = df['lon'].apply(lambda x: "MO" if x < -91 else "MD" if x > -85 else "MS")
```

Now, one-hot-encoding can be done with this new categorical variable.

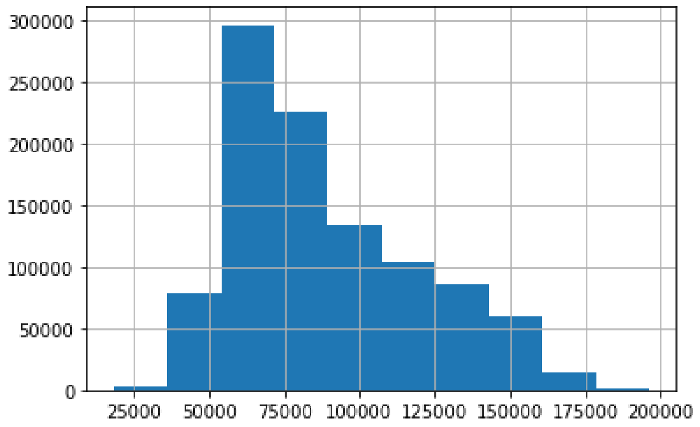
```
In [17]: df[['MD', 'MO', 'MS']] = pd.get_dummies(df['Region'])
```

The last operation is to possibly standardize or normalize any variables in the dataset. This is needed because many models will work with numerical data that are in different scales. By taking the numerical variables in this dataset and converting them into a single scale, the models that are used can make more effective predictions about the GPA.

Two variables that are in need of standardizing or normalizing are **year and salary**. This is because the values for these variables are far greater compared to the other data points. To determine if we need to standardize or normalize, the distribution and range of values for each needs to be examined.

```
In [18]: df['parents_income'].hist()
```

```
Out[18]: <AxesSubplot:>
```



The histogram for the parents income has a positive skew, but the range of values is from roughly 25000 to 200000. This is a small range of salaries. **Since there are no extreme salaries, like those in the 1 millions, normalization is a good choice here. So, the 'parents_income' column will be normalized, using min-max normalization technique.**

In [19]:
```
df['parents_income'] = (df['parents_income'] - df['parents_income'].min())/ (df['parents_income'].max()- df['parents_income'].min())
```

Since the 'year' category also follows a short range of values, this column can be normalized using the same approach above.

In [20]:
```
df['year'] = (df['year'] - df['year'].min())/ (df['year'].max()- df['year'].min())
df['year'].head(5)
```

Out[20]:
```
0    0.10
1    0.15
2    0.15
3    0.20
4    0.20
Name: year, dtype: float64
```

The data has now been cleaned. The next sections will discuss how the models will be evaluated and will show the results of these models when they predict the students' GPAs.

Evaluation

Explanation & Ideas

To evaluate this model, there are many options to consider. Since GPA is being predicted, the goal is to ensure that the predicted GPA is close to the actual GPA. So, the amount of error needs to be reduced as much as possible between the predicted GPA and the actual outcome. For this reason, smaller errors are favored to larger ones, so penalizing larger errors is important when evaluating this model.

It is important to address that this dataset includes of missing data (as seen with the tutoring section) and contains outliers for the avg_hours_studied section. The outliers shouldn't be of much concern, since we trimmed data entries that had 10,000 average study hours. Another challenging aspect is that the data does have clusters when it comes to the longtitude and latitude. This needs to be accounted for this if the location is chosen as a potential feature/independent variable for the regression models to predict GPA.

For the reasons mentioned above, **one metric to evaluate the performance of our model** is the **Mean Squared Error, or MSE** . One important aspect of this metric is that it is sensitive to errors, so larger errors will be penalized way more than smaller ones. Because it increases larger errors, MSE is sensitive to outliers. A high MSE will indicate poor performance of the models, so a small MSE will be indicative of a successful model.

To evaluate the model, **10-fold cross-validation** is used. By dividing the data into folds and testing the performance of the model on these folds, improve the generalizability of the model is improved. This also prevents it from being very overfitted with the training data.

What To Be Aware Of: As mentioned, the dataset has clusters for the location. While techniques like spatial cross-validation can be used to deal with these clusters, it is important to see if the location has a direct impact on the GPA. If it does not, then it is not need to do spatial cross-validation. In addition, the range of GPAs the regression model outputs needs to be considered. If the model outputs a negative GPA or one that is beyond a 4.0 scale, that could be a potential issue as well.

Modeling

KNN: Numerical data, such as the avg_hours_studied and age of the students to work with, is provided. Categorical variables have also been converted into numerica data via one-hot encoding. Using the sklearn.neighbors library, the KNN model for regression can be run. The sklearn.model_selection library provides a way to evaluate the model with cross validation.

In [23]:
```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import cross_val_score
```

For KNeighborsRegressor, we need to select a value for k (n_neighbors) which is one of the hyperparameters of this algorithm.

Choosing a value of k is a bit arbitrary and there are is no set technique to do so. However, one approach we can try is to take the sqrt(k), which is commonly done. Here is the link to the article that mentions this idea: LINK . Since our dataset has nearly 1 million entries, k will be sqrt(1000000), which is 1000. This is a large k value, and this may lead to the evaluation of the model taking a long time.

In terms of features to use , avg_hours_studied, parents income, and credits will be used. These are are all numerical variables, which work well with KNN. One important idea to note is that parents_income has been normalized to a 0 to 1 scale in order to deal with its larger values. Both avg_hour_studied and credits however, remain the same.
In terms of k, selected a k-value of 1000 has been chosen. This is the sqrt of 1,000,000 which was the size of the original dataset. With the features selected and hyperparameters in place, the model can be run and evaluated:

In [28]:
```
x = df[['avg_hours_studied', 'parents_income', 'credits']]
y = df['gpa']
knn = KNeighborsRegressor(n_neighbors=1000)
```

```
scores = cross_val_score(knn, x, y, scoring ='neg_mean_squared_error',cv=10)
scores.mean()
```

Out[28]:
```
-0.14131233414734257
```

<u>After running for a long time, we have obtained an MSE of 0.1413.</u>

<u>The next regression model that we will use is the Decision Tree Regression:</u> Since the data contains categorical information like the major of the student and the region they are from, a Decision Tree will be a helpful model as it works well with categorical information. Furthermore, the decision tree algorithm can help determine the important features to work with, making this useful for predicting GPA.
<u>One hyperparameter to consider is the depth of the tree</u>. Since there is no set strategy to determine the right depth for the decision tree, a depth of 20 is chosen. This will allow us to work with our large dataset but also choose the importance of the features that are being provided to the decision tree regression. In addition, with this depth, the model will not overfit the data and make proper generalizations from it.

In [21]:
```
from sklearn.tree import DecisionTreeRegressor
```

In [24]:
```
# using features: parents_income, major, avg_hours_studied and tutoring
x = df[['parents_income','CS', 'Engineering', 'English', 'Math', 'Physics', 'avg_hours_studied', 'No', 'Yes']]
y = df['gpa']
dtree = DecisionTreeRegressor(max_depth=20)
scores = cross_val_score(dtree, x, y, scoring ='neg_mean_squared_error',cv=10)
scores.mean()
```

Out[24]:
```
-0.19668026525850246
```

From the decision tree regression, we obtained a result of 0.1966. This is a bit higher compared to the KNN model. Let's observe if this model's performance will be enhanced if categorical features are only provided to this model. The code for this is listed below:

In [25]:
```
# Remove numerical features (parents_income and tutoring)
x = df[['CS', 'Engineering', 'English', 'Math', 'Physics', 'No', 'Yes']]
y = df['gpa']
dtree = DecisionTreeRegressor(max_depth=20)
scores = cross_val_score(dtree, x, y, scoring ='neg_mean_squared_error',cv=10)
scores.mean()
```

Out[25]:
```
-0.3662638838000397
```

By using only categorical features, the decision tree model has outputted an MSE of 0.366. This is larger than the error produced previously. This could serve as an indication that parents_income and avg_hours_studied are features that can improve the performance of this model.

<u>Last, a simple Linear Regression Model is used:</u> This model is relatively straightforward, and it will be interesting how the inclusion of categorical variables like major, tutoring, and the region of the student are utilized will impact this model's performance. It is also interesting to see if variables directly associated with GPA, such as avg_hours_studied and parents_income, will optimize this model's performance. However, given that this model is relatively simple, a very low MSE is not expected.

In [27]:
```
from sklearn.linear_model import LinearRegression
linear = LinearRegression()
x = df[['parents_income','CS', 'Engineering', 'English', 'Math', 'Physics', 'avg_hours_studied', 'No', 'Yes', 'MD', 'MO', 'MS']]
y = df['gpa']
scores = cross_val_score(linear, x, y, scoring ='neg_mean_squared_error',cv=10)
scores.mean()
```

Out[27]:
```
-0.20906545328492182
```

As expected, this model produced one the higher MSE's. However, it did run very fast compared to the Decision Tree and KNN Regressor models. Similar to the decision tree model above, it will be interesting to see how this algorithm would perform if it only used categorical variables as opposed to numerical ones. The code for this is displayed below, with the following features:

In [28]:
```
# testing model with only categorical features
x = df[['CS', 'Engineering', 'English', 'Math', 'Physics','No', 'Yes', 'MD', 'MO', 'MS']]
y = df['gpa']
scores = cross_val_score(linear, x, y, scoring ='neg_mean_squared_error',cv=10)
scores.mean()
```

Out[28]:
```
-0.3662765443716744
```

Once again, the model has produced a higher error when only categorical features are used for predictions. Given more time, it would be appropriate to revisit the data cleaning process and see how the features can be re-engineered such that they reduce the MSE of the models.

Conclusion

For the models, I used **KNN, Decision Tree, and Linear Regression** , and I utilized MSE to evaluate them. Out of all of these models, KNN produced the smallest MSE, which was 0.141. The next best performing model was the Decision Tree Regressor (MSE of 0.1967) and finally the Linear Regression, which was 0.209. **In terms of success, I do not think my models were very successful.** Since I am working with GPA, it's important to reduce the MSE as much as possible, but with MSE's like 0.1967 and 0.141, its clear that the models weren't exactly the best at predicting GPA. A GPA that is a 3.7 is quite different from 4.0, so I my models could have definitely performed more better.
During this project, one interesting challenge was determining what features would work best for the GPA prediction model. I got a sense of this in the initial part of this project, when I explored the data and found results from hypothesis tests. Of the features available, I found that features such as **avg_hours_studied, parents_income, and the major** of the students were the ones I mainly utilized in my models. For the data cleaning step, I began with dealing with any missing values, which the 'tutoring' column consisted of. After, I removed the outliers that were present in the avg_hours_studied section. I also modified the long and lat variables by defining cluster groups and categorizing the entries in each row into their respective clusters, which represented a particular region in the United States. Finally, I one-hot encoded any variables. After this data cleaning process, I had to select the necessary features and adjust the hyperparameters for my models before evaluating them.
Going forward, additional data that would be helpful would be the hours of sleep a student gets, the number of extracurricular activities they are

involved with, and a categorical variable related to their mental health. I think these variables could be directly related or influence a GPA, which is why I think this data would help improve the performance of models.