SRP II - Project Journal

Brian LaMons

I. 9.24.24-9.25.24 Entry: Preliminary Analysis of Kepler Transiting Data

A. Searching for Data & Initial Interpretations:

By utilizing the exoplanet database on the Caltech website (linked here), I was able to locate a variety of transiting light curves (located at the following linked archive) produced by the Kepler and K2 missions. Hopefully, I will be working with NASA later this year (beginning in January 2025) in an internship on sorting through and analyzing larger Kepler and TESS datasets for the purpose of discovering new exoplanets.

1. In preparation for that, after locating the database, I opened a folder containing light curves which consisted of .fits files. This file type is designed specifically for containing astrophysical data, and has to be processed in a certain software produced by the Smithsonian Astrophysical Observatory (SAO).
2. Once downloading this software and opening one of the light curve files in it, the software processed the data and produced the following image:

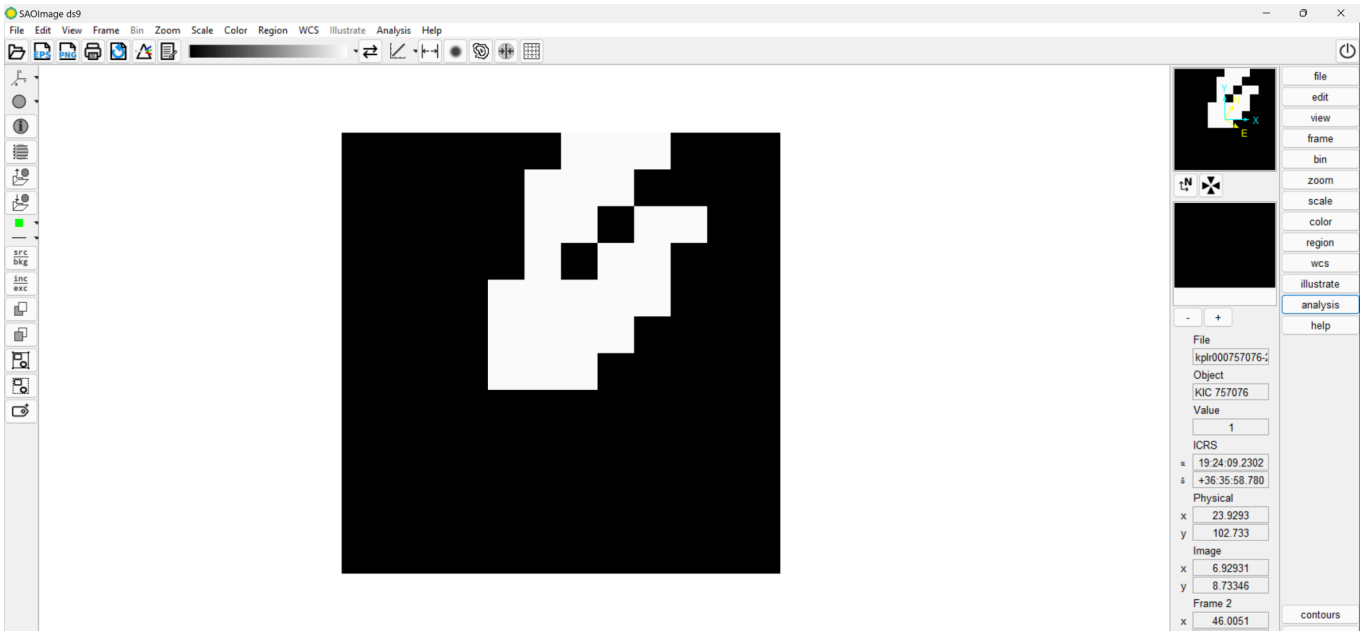


Figure I.1 - SAO-processed image of Kepler transiting data; black-colored squares are a lack of light, while white-colored squares are detected light from a host star.

3. The SAO software can also be used to produce a graph from the above image, that graph being the typical “light curve” one would expect from this data:

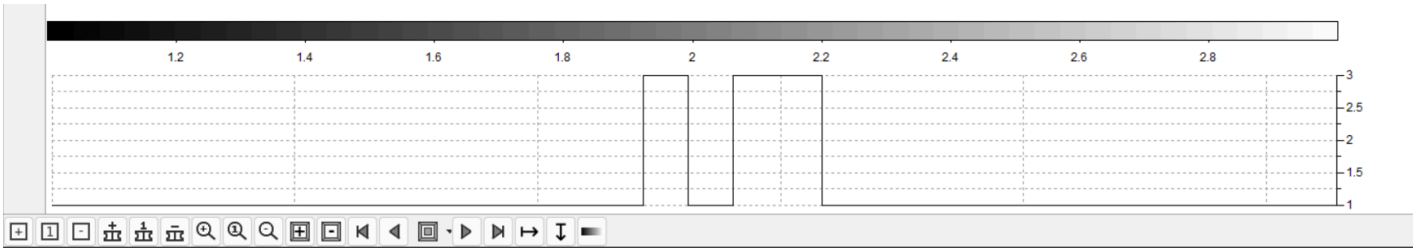


Figure I.2 - SAO-produced graph of absorbed light versus time (of the observation period, in days); the two rectangular increases in the graph represent an object passing in front of the host star, possibly an exoplanet.

B. Calculation of the Distance from the Host Star of the Given Object:

1. At this point, now that I had the data necessary to begin the calculation, I determined the distance between the object represented in figure I.2 and its host star using Kepler' Third Law of Planetary Motion.
2. However, I first had to determine the object's orbital period, or the distance between transits (dips in the light before utilizing Kepler's Law. After taking additional graphs of the object's data in the SAO program (see figures I.3-I.4), I determined its orbital period to be approximately 0.2 days. This is due to the fact that, in all three light curves taken from the data, the two shifts in absorbed light are approximately 0.2 units away from one another on the x axis.

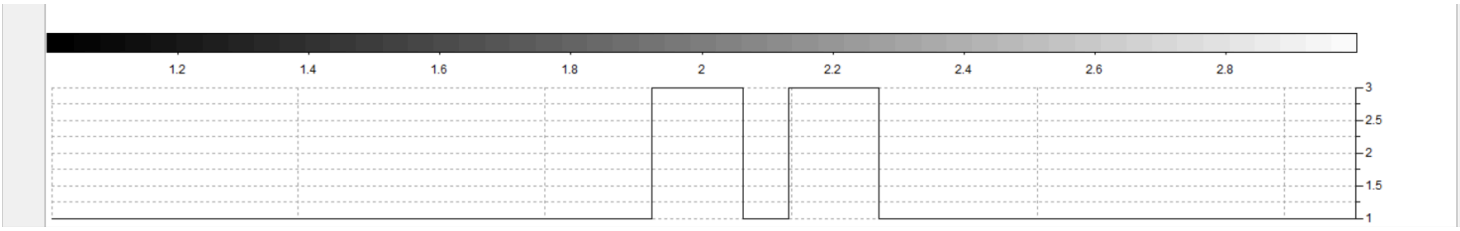


Figure I.3 - Light curve for the same object as represented in figures I.1-I.2, however, this is for the subsequent transit data point.

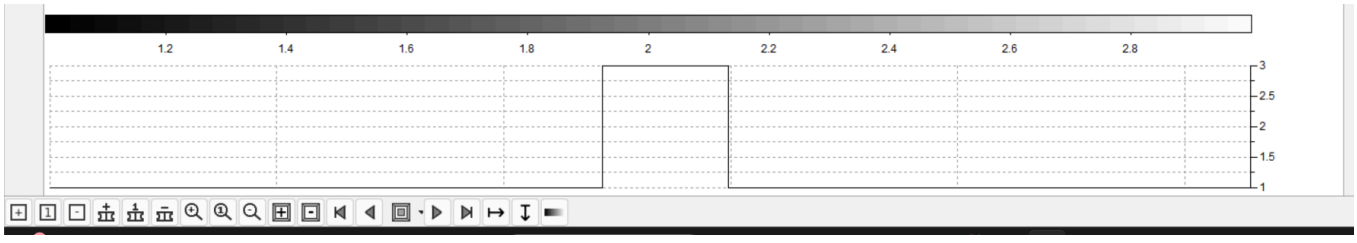


Figure I.4 - Another light curve for the same object, but for the third transiting data point.

3. Then, with Kepler's 3rd Law, which is defined as P^2 = a^3 (where P = the orbital period and a = the average distance between the host star and the object, or the major axis), I determined a to be about 0.585 astronomical units (au) ([0.2]^2 = a^3). As one au is about the distance between the Earth and the sun, this possible exoplanet is a little more than half that distance from its own parent star.

C. Goals for the future:

1. In the coming days, I hope to continue with this analysis by determining both the size of the object using the ratios that can be produced from the above graphs and the "wiggle" of the possible exoplanet in reference to its host star. That way, I can continue to practice and hone my skills at performing analysis with Kepler data in preparation for the internship or the project that will follow.

II. 10.5.24-10.9.24 Entry: Secondary Search of Databases & Attempts to Verify Data

A. Second Database Search Trial (using Kepler-10):

While on the same database as mentioned above (i.e., the Caltech-NASA exoplanet archive), I consulted the website's user manual in order to determine how best to make a search and locate the data of a specific object.

1. The data corresponding to a certain object can be found using a *Time Series Lookup by ID* kind of search. This variant of search applies when data is taken in time series, as was done in the Kepler and K2 programs, and when one has the KIC number (ID) of a certain object.
2. To test this out, make a *Time Series Lookup by ID* search for Kepler-10, which has a KIC number of 11904151. That search produced the following results, as shown in Figure II.1.

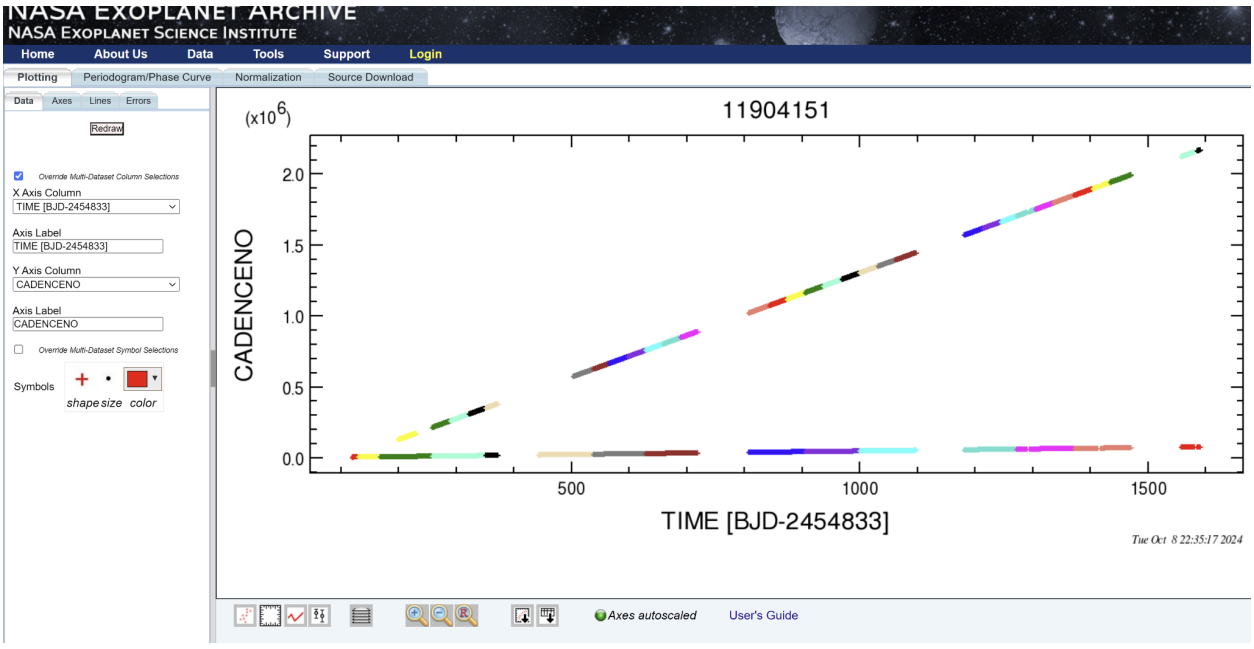


Figure II.1 - The initial screen produced when one enters in an ID in the time series search section of the database. From this screen, you are able to navigate to a number of different tabs and produce a variety of graphs depending on your desired parameters.

3. As seen above, by this point, I had successfully located the time series data corresponding to Kepler-10, however, I had not yet found a light curve for said object. In an attempt to find such a light curve, I utilized the "phase curve" function seen on the top left corner of Figure II.1, as a phase curve is a graph of the change in brightness of a star-planet system during a planet's orbital period (this is very similar to what I am looking for in a light curve).
4. In order to make a phase curve from the given data (recorded in the table on the right-hand side of Figure II.2) I was prompted to select data points, which I did so to test out how the system functioned.

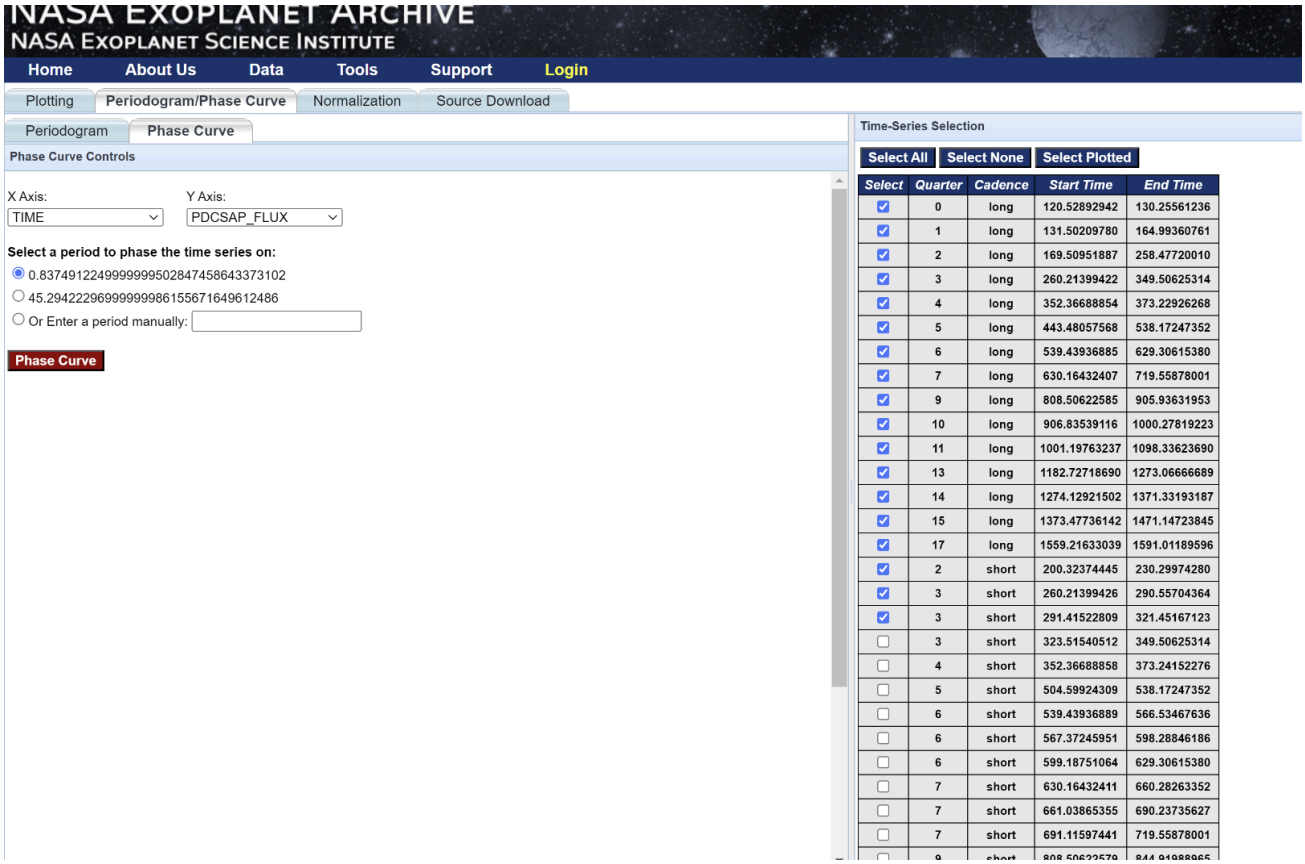


Figure II.2 - The initial screen where one selects the parameters for creating a phase curve from the collected data. For testing, I only selected about half of the total number of data points, simply to see what the program produced.

5. After shifting the parameters of the y-axis several times, I produced the following graphs (Figures II.3-4). The first one plots cadence (events evenly spaced within time) versus phase, while the second plots flux (in this case, the amount of light arriving from the star) versus phase. It appears that the second graph (Figure II.4) is very close to what I am looking for, as it presents the change in light intensity over time, albeit in a different format than initially expected.

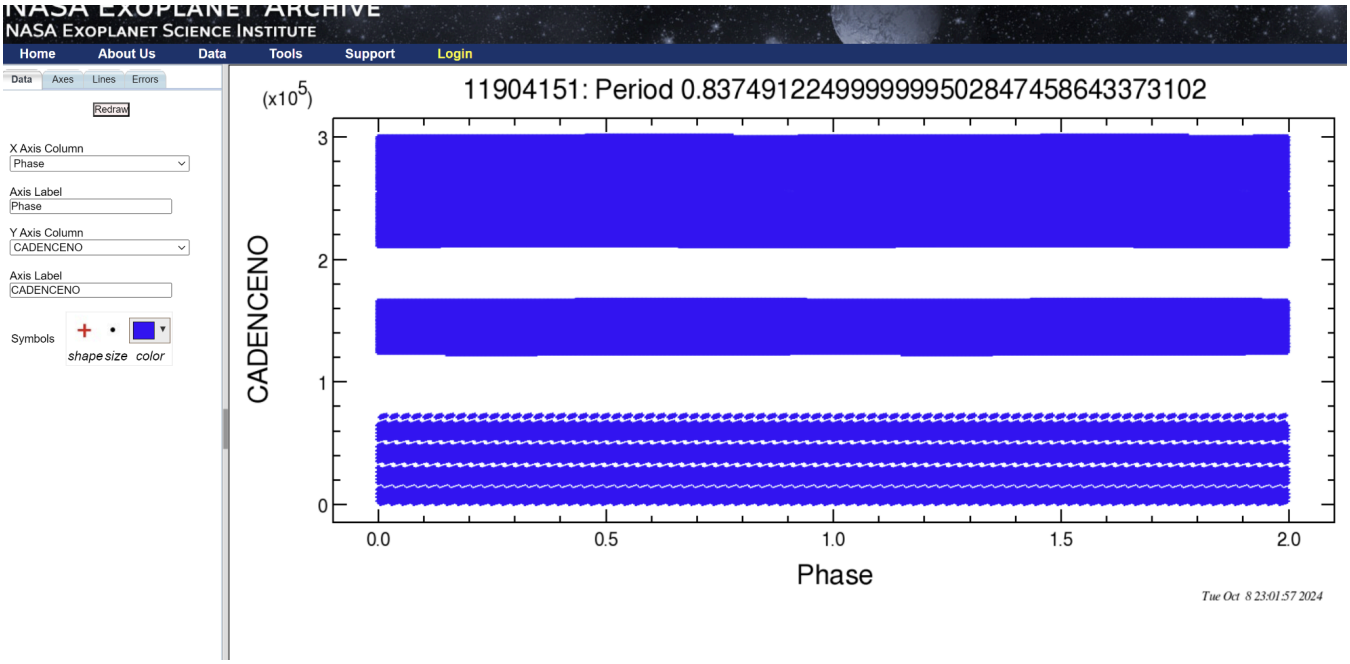


Figure II.3 - The Phase vs. Cadence graph; this graph does not appear to provide any valuable or useful information to my search in particular, however, it does indicate how and at what intervals the data was collected.

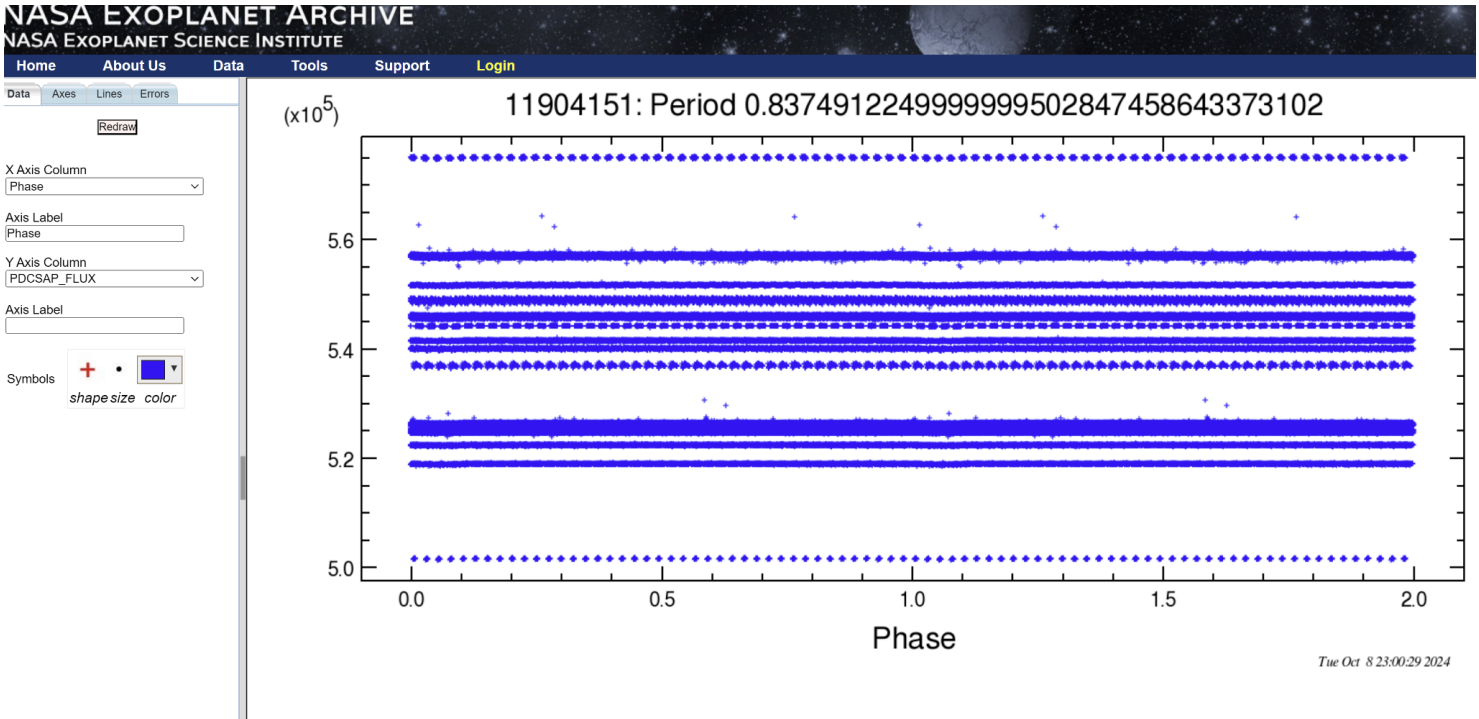
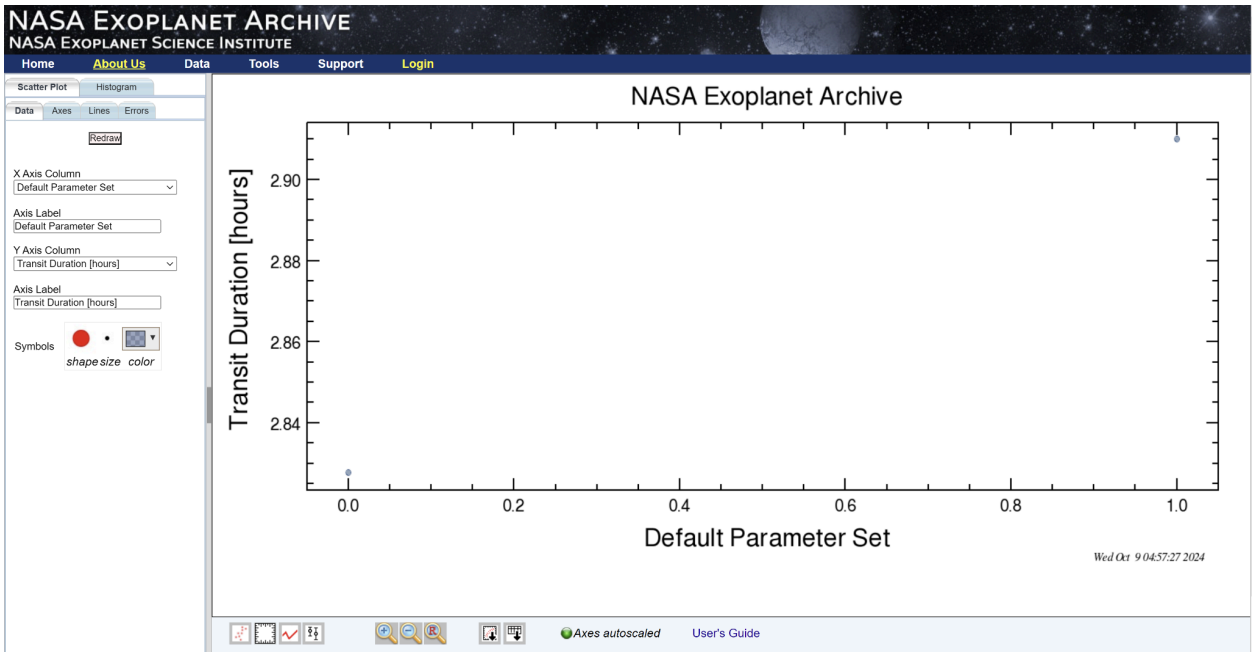


Figure II.4 - The Phase vs. Flux graph; this graph shows where flux, or total light arriving from a star, rises and falls, somewhat similar to what a light curve shows. However, it is not nearly as clear as a light curve, and did not satisfy my search.

B. Third Database Search Trial (with TESS object HD 110082 b):

Due to the fact that my second search trial was largely unsuccessful at obtaining a light curve, I decided to try a different method of searching using the TESS confirmed exoplanets database. Beginning from the Exoplanet Archive homepage, I looked into TESS transiting data, and, utilizing the user manual, I realized that I could plot certain pieces of data related to one TESS object from the table given by eliminating or selecting certain filters.

1. Thus, by eliminating all filters except for the transiting period, I produced the following plot (Figure II.5), and although it is not yet a light curve, I believe that, with further manipulation of the filters, I will be able to produce a light curve plot with this function.



C. Plans for the Future:

1. For the next cycle, I will attempt to manipulate the plot to produce a light curve, as I now have the requisite knowledge to do so from my two trials and my utilization of the user guides throughout the Exoplanet Archive website.

III. 10.15.24-10.23.24 & 11.2.24-11.5.24 Entry: Parallel Trial of Exoplanet Databases for Light Curves

A. Goals:

Although the previous searches appear to have led to a light curve (as exemplified in the zoomed-out graph that is figure II.3, of flux versus the period in days), after discussing with Dr. Matone, we agreed to retry the attempt to find a light curve using two databases: the NASA Exoplanet Archive, which has been utilized prior, and the Mikulski Archive for Space Telescopes, linked here (also see image below of the MAST data portal in figure III.1). As both databases contain TESS objects, I aim to select one given TESS object and attempt to find the corresponding light curve from both databases.

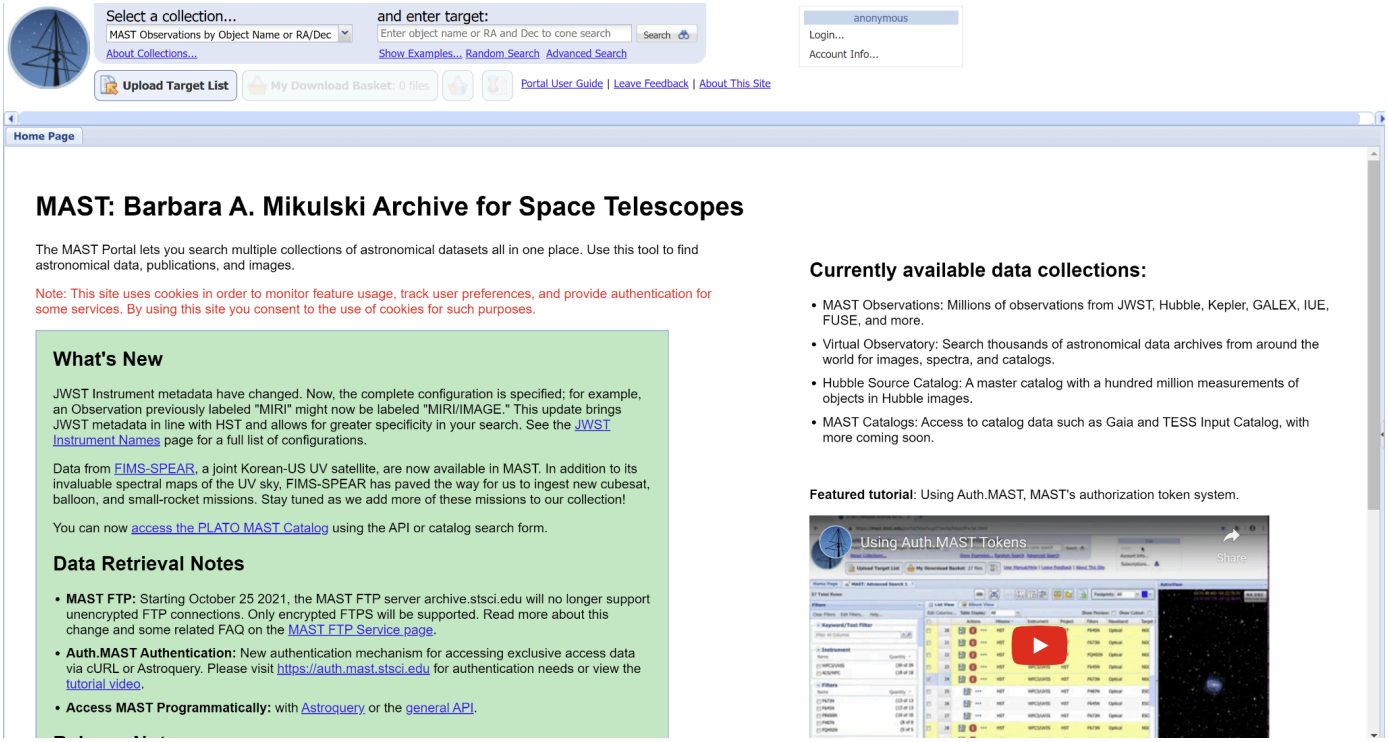


Figure III.1 - The home screen of the MAST data portal.

B. Education on Utilizing the MAST Data Portal & Application to Test Case:

1. In order to begin using the MAST data portal for this parallel study, I first had to complete an initial search and plot to acquaint myself with the workings of the platform. Thus, I plotted an example TESS light curve (or a test case) by following the instructions and guidelines provided within the following GitHub file.
    a) For this example, which plots the light curve from object WASP-126 b (which is, in fact, a confirmed exoplanet), I began coding in Windows Notepad++, storing the code file within my

Google Drive. As the code is in python, I had to install a certain plugin (specifically, the NppExec plugin) so that I would be able to run the code. However, after attempting to run the code from Notepad++, I realized that there was an issue with the plugin itself, and so I decided to use Thonny Python IDE instead.

b) Once I re-wrote the code within Thonny and once again, attempted to execute it, it returned a fairly major syntax error (see the bottom of figure III.2). In order to fix this issue, I researched a variety of resolutions, none of which seemed to improve the function of the code.

Figure III.2 - IDE view of the light curve graphing script in Thonny python. The comments above each portion of code indicate their role in the overarching process of extracting the file from the database, downloading it, and plotting a variety of points from the file onto the graph that should be produced. However, as indicated by the error at the bottom of the figure, the code did not work properly in Thonny.

Nevertheless, during that research, I learned that the first line of code is written in the syntax of a certain command that only functions within a Jupyter notebook file. Thus, I re-entered the

code into Jupyter (see figure III.3), which I have experience using from my SRP I project.

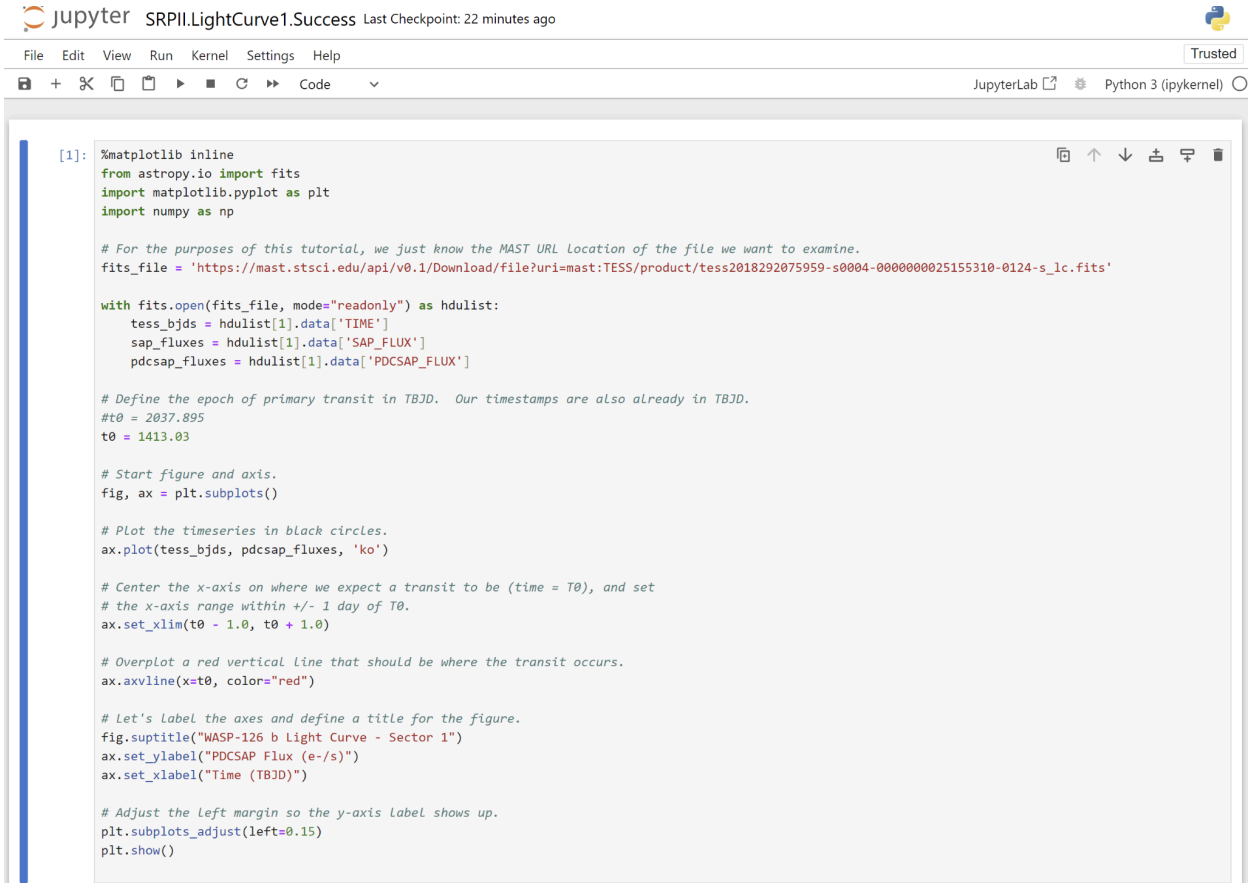


Figure III.3 - The same code as above, except that here, it is seen in a Jupyter notebook file. The syntax used, notably in line one, is specifically designed for a Jupyter notebook, and thus the code was able to work as intended in this editor.

c) Upon running the cell of code in Jupyter, the code functioned as intended and produced the following light curve (see figure III.4) of the planet WASP-126 b, which orbits the star TIC 25155310:

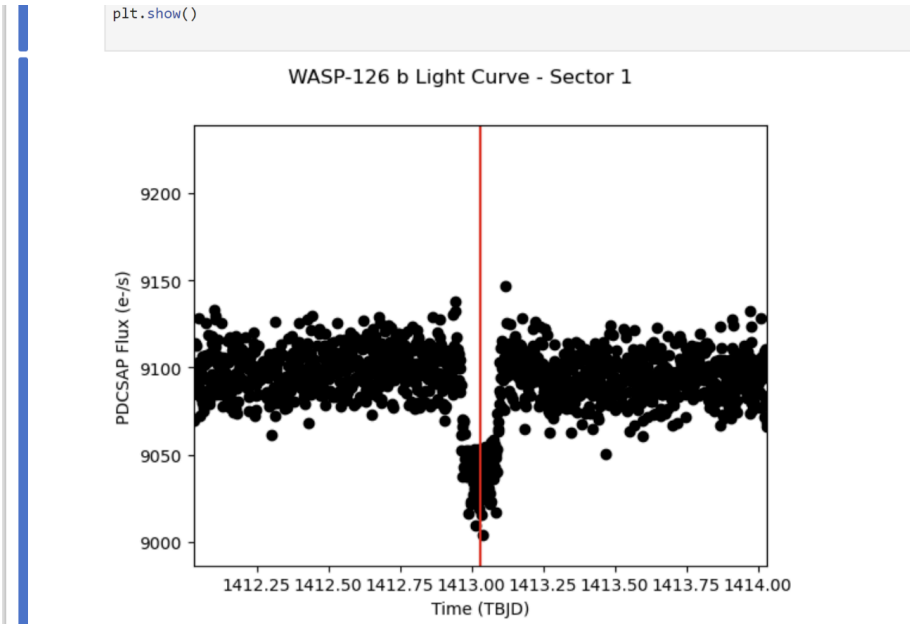


Figure III.4 - This is the light curve graph (PDCSAP flux, or instrumentally corrected brightness, vs. time in dates corrected for non-Earth measurements)

produced from the above code in Jupyter.

d) During this process, I also learned how to interpret the somewhat puzzling diagrams produced in my initial attempts at finding a light curve in entry I (see figure I.1). This is known as an "aperture pixel diagram," and its values, when broken down into powers of two, indicate that a given pixel was perceived by the spacecraft and used to calculate different data points, such as the PDCSAP flux c (which is shown in the graph above, figure III.4). Note the PDCSAP flux refers to the simple aperture flux (SAP) corrected for instrumental variations, making the PDCSAP flux the best estimate for the brightness of the target star.

C. (**11.2.24**) Applying the Code to another TESS Object from the MAST Database:

1. Now that I have learned the requisite skills and operating parameters for extracting and producing light curves from the MAST database, I will repeat the above process for a TESS object that also appears in the NASA Exoplanet Archive. I selected that object as **HD 108236 b**, which is a confirmed exoplanet.

```
# Define the epoch of primary transit in TBJD.  Our timestamps are also already in TBJD.
#t0 = 2037.895
t0 = 1413.03
```

Figure III.5 - Close-up snapshot of figure III.3 that showcases the role the epoch of primary transit (midpoint of light curve) plays in the production of the light curve. The t_0 value serves as the center of the sample light curve that is figure III.4.

2. Before I do so, there is a necessary fact that I must be able to figure out about any given exoplanet and its transiting data: this is the *Epoch of primary transit*, or the midpoint of a given transit. By knowing this, which is marked as t_0 in the above code (see zoomed-in image above in figure III.5), one is able to clearly find the minimum of (i.e., the center of) the light curve for a given exoplanet.
   a) At first, I thought that one could determine this fact mathematically, and though that is possible, it requires knowledge of either the expected midpoint (epoch) or the precise midpoint. Without at least one of these, one is unable to solve the equation

below (figure III.6) for the midpoint and thus plot the light curve from its minimum, as was done above in figure III.4.

4. Next, the ephemeris is computed:

- *Transit Ephemeris Method*: Since the transit midpoint is known, the algorithm simply iterates over the period, including uncertainties, such that

$$T_{transit} = T_{knowntransit} + iP$$

Figure III.6 - The above equation can be used to calculate the midpoint of a transit (and thus the midpoint of a light curve), where $T_{transit}$ *is the time of the predicted midpoint,* $T_{knowntransit}$ *is the time of the known midpoint,* i *is any integer, and* P *is the orbital period.*
– Source: [NASA Exoplanet Archive - Transit Algorithms](#).

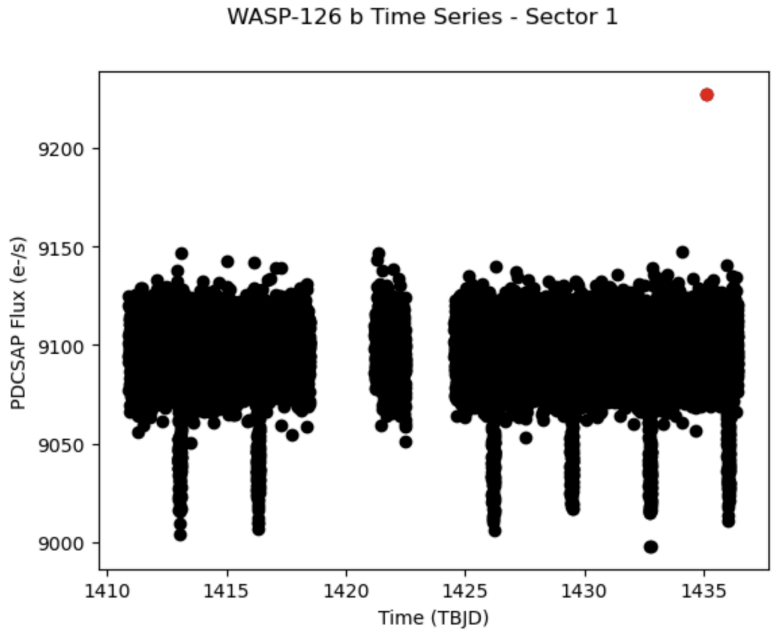
b) With that first option exhausted, I turned to a different, more "top-down" approach to the data. With this method, one must first locate the whole series of data and place it on a plot before finding the midpoint (epoch). Otherwise, one is simply guessing on where the exact data points are located and fails to achieve the goal of producing a light curve efficiently.

c) Therefore, to first plot the time series of a transit (which represents the entirety of the transit data collected by TESS from the beginning of observation to its conclusion), I had to implement several new lines of code, the basis of which I was able to find within the previously used GitHub file. That code is seen below in figure III.7.

```
with fits.open(fits_file, mode="readonly") as hdulist:
    qual_flags = hdulist[1].data['QUALITY']

# Start figure and axis.
fig, ax = plt.subplots()

# Plot the timeseries in black circles.
ax.plot(tess_bjds, pdcsap_fluxes, 'ko')

# Locate quality flags greater than zero.
where_gt0 = np.where(qual_flags > 0)[0]

# Overplot the fluxes with quality flags greater than zero in red.
ax.plot(tess_bjds[where_gt0], pdcsap_fluxes[where_gt0], 'ro')
```

Figure III.7 - Additional code required to produce a graph of the whole time series (all detected transits). The "qual_flags" variable serves as the means of extracting the beginning and end limits of the data, or its "flags."

Figure III.8 - Time series for the above sample object, WASP-126 b, produced with the code above. It showcases all of the transits detected, one of which is focused upon to create the light curve in figure III.4.



d) Finally, now that I had plotted the time series, I was able to see where the most significant minimums in flux occurred. Those minimums are the primary transit epochs, and by entering one of their values as t_0 in the original code, I was able to produce a centered light curve for a second object, **HD 108236 b** (see figure III.11), from its time series (see figure III.10).

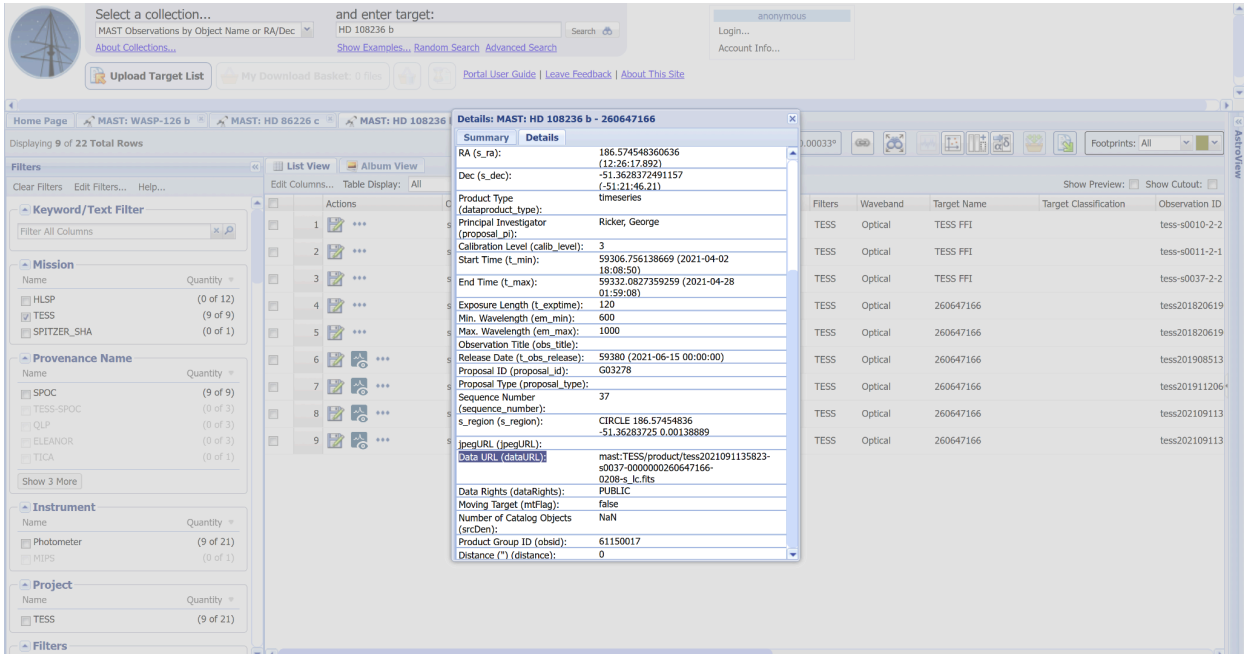


Figure III.9 - View of the MAST search window by which I found the data for HD 108236 b. For the code I utilize, the most important piece of information is the data URL, which is how the computer finds the data to plot when the code is executed.

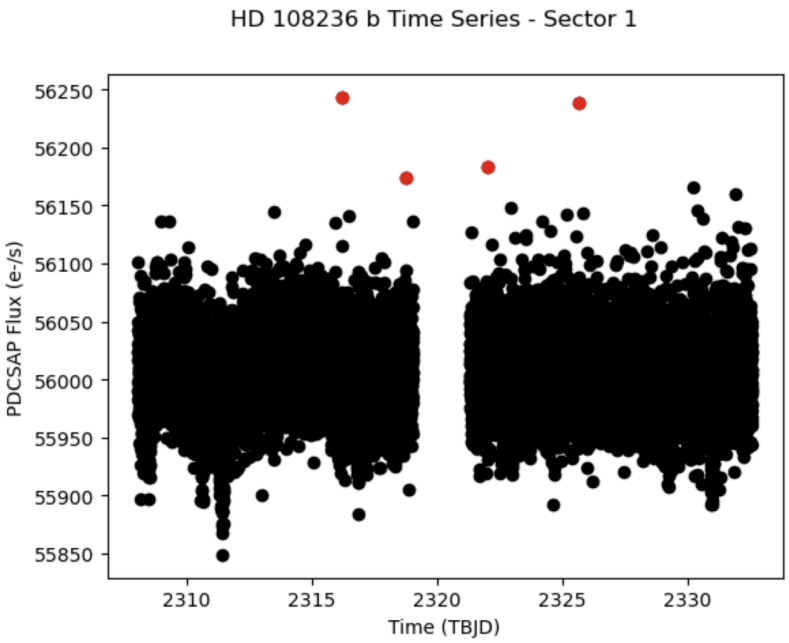


Figure III.10 - Time series for the exoplanet HD 108236 b, as produced with the code discussed above. Values taken from the MAST database.

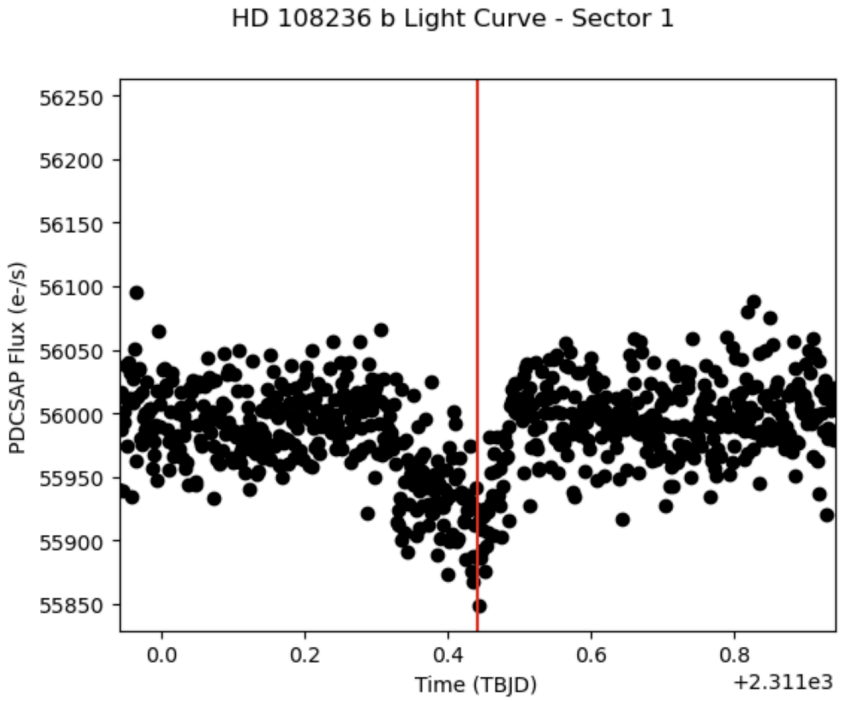


Figure III.11 - The centered light curve created from the left side of the above time series. The primary transit epoch in this case is equal to approximately 2311.44 BJD (adjusted Julian calendar days).

D. (11.3.24) Calculating Exoplanet Parameters from HD 108236 b Light Curve & Comparison with Parallel Database:
   1. Using the light curve above (specifically, the width of the central dip along the x axis), I determined the transit period for HD 108236 b to be about 0.11 BJD, or 2.64 hours.
      a) I then decided to verify this result with the NASA Exoplanet Archive, which also hosts data on HD 108236 b; according to their records, its transit period (or duration) is about 0.1025 BJD or 2.46 hours. Therefore, my calculation based on the above light curve has a percent error of 6.82%.
   2. Subsequently, I found the orbital period (P) of HD 108236 b to be about 3.5 days by looking at the x-axis distance between transits on the time series graph (figure III.10). Again, comparing this to the NASA Exoplanet Archive value of 3.79, my number has a percent error of roughly 8.29%.
   3. Then, with Kepler's 3rd law, or $P^2 = a^3$, where P is the orbital period and a is the size semi-major axis of the orbit (distance from host star to exoplanet), I determined the distance between HD 108236 b and the star it orbits. $(3.5)^2 = a^3$, so a = 2.31 AU. Therefore, HD 108236 b is a third more than twice the distance between the Earth and the Sun from its host star.

E. Summary of Parallel Study:
   1. What has been accomplished:
      a) I now have the knowledge necessary to code, within a Jupyter notebook file, the extraction of transit data to produce a light curve from the MAST database.
      b) Further, I have, by learning how to find the time series and thus the primary transit epoch, universalized that knowledge to all TESS objects in the MAST database.
      c) From that light curve, I have calculated not only the transit period, but also the orbital period and distance from the planet to its host star.

IV. 11.15.24-11.18.24 Entry: Initial Research into NASA Deep Learning Tools & Applications

A. Scikit-Learn Introductory Notes:

1. One of the two tools NASA utilizes for deep learning analysis of light curves like those seen above, Scikit-Learn is a python machine learning library used most often to classify data. To be precise, NASA uses it to determine if collected data is, in fact, an exoplanet rather than smaller astrophysical phenomena (like asteroids, for instance) by training it on data that has already been confirmed as indicative of an exoplanet.
2. Installation Process:
    a) In order to begin working with this software, it was necessary that I install it, which I did so using the command prompt and pip (see figure IV.1) according to the linked documentation from their website. Although I originally encountered a syntax error when making the attempted installation, further research (as well as additional trial and error) indicated that this error was a result of my having typed 'python' rather than 'py.'

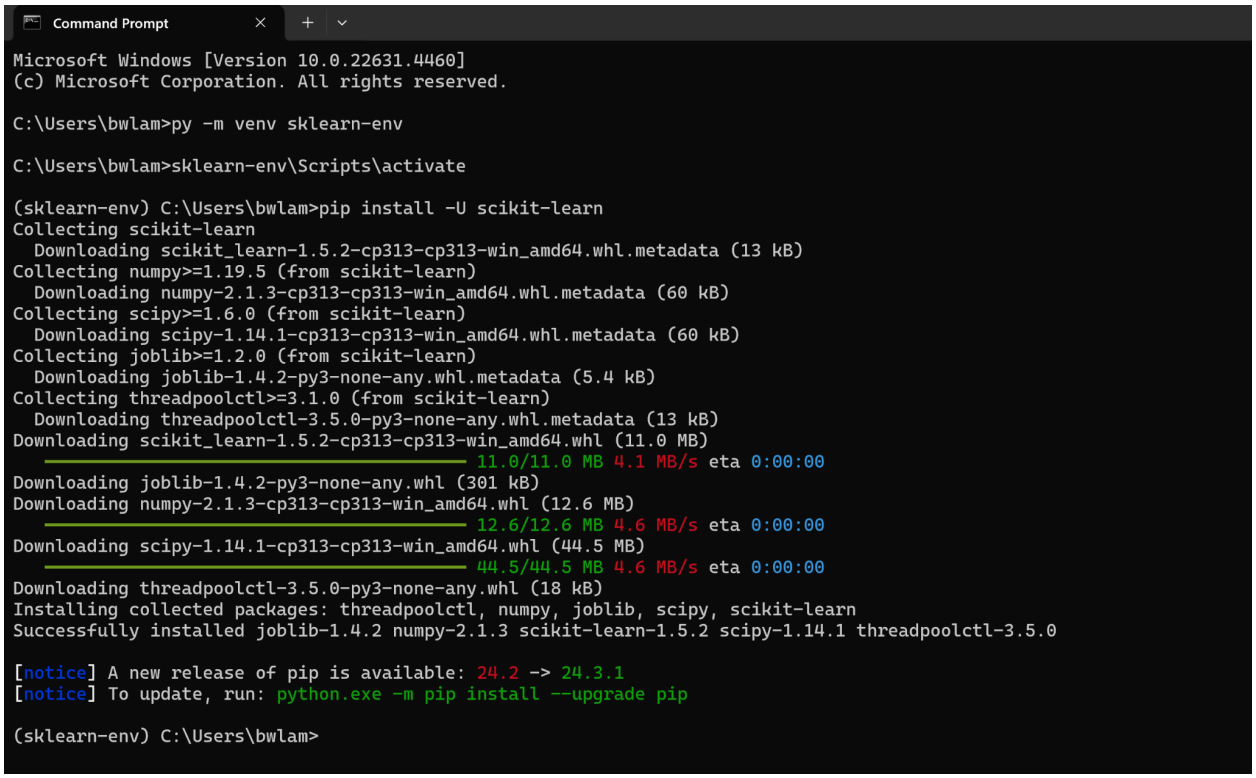


Figure IV.1 - Code from the command prompt used to produce the virtual environment and install Scikit-Learn. The fact that I am typing in a virtual environment is indicated by the environment name in parentheses to the left of the commands.

b) Additionally, I decided to install Scikit-Learn within a python 'virtual environment' that I named 'sklearn-env,' which means that the program was installed in an isolated location rather than globally. This is done so that no other python packages are inadvertently impacted in the future.

(1) Virtual environments do save on one's computer, so one should remember to **activate** it (with the 'sklearn-env\Scripts\activate' command) when beginning work on the project and **deactivate** it (simply with 'deactivate') when done for the moment.

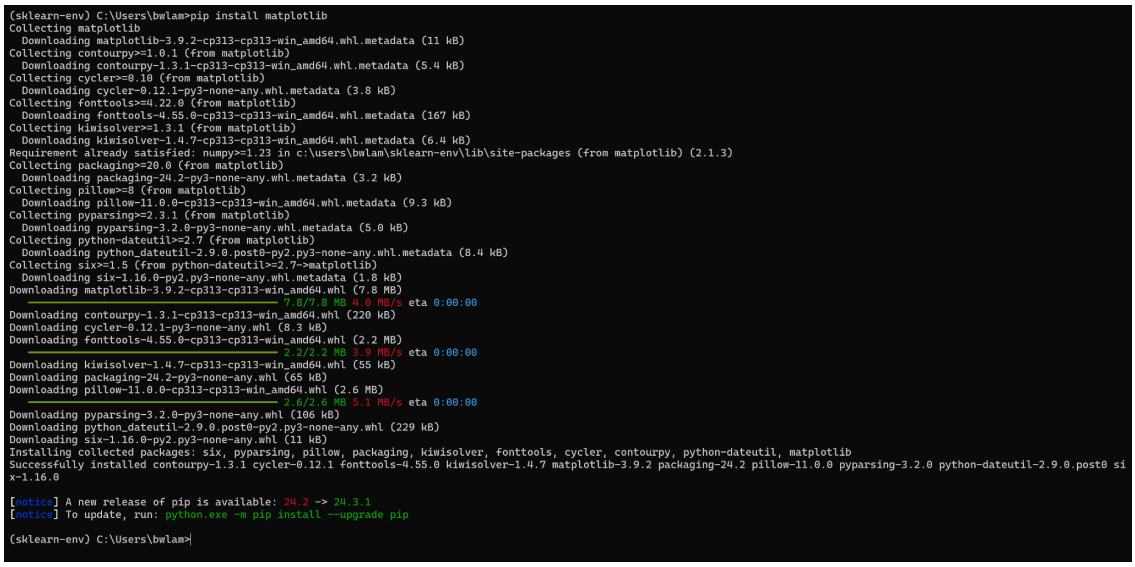


Figure IV.2.1 - Code used to install the matplotlib library, which is designed to produce charts, graphs, and images from machine learning data and results.

c) Plus, as seen in figures IV.2.1-2, I needed to install both the package matplotlib (so that I would be able to plot data and test it with a wider variety of tools) and the package pandas (required to run additional benchmarks and tests).

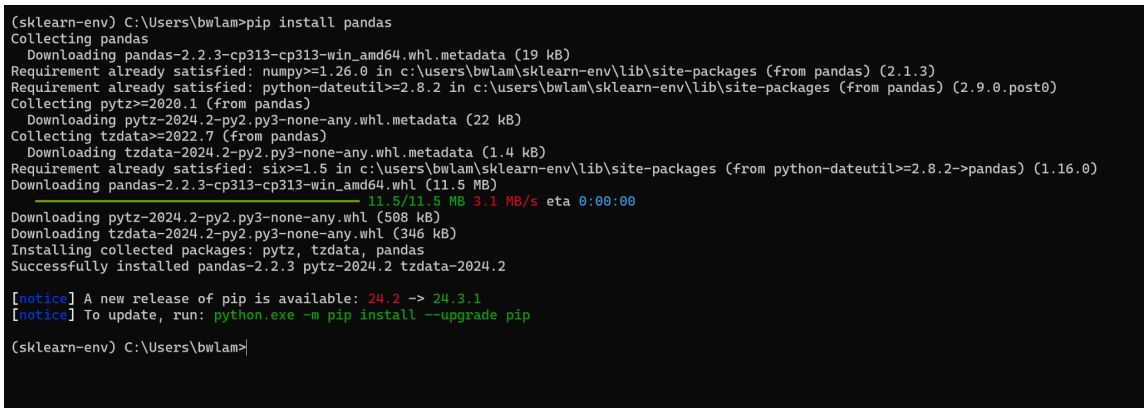


Figure IV.2.2 - Code that allowed for the installation of the pandas machine learning library.

3. (REFERENCE) Brief overview of General Commands & Examples:
    a) I then took a look at the official Scikit-learn documentation to educate myself on some of the commonly used methods for machine learning with this application. My notes are below:
        (1) Data is organized into *samples*, which have several *features* (if they have more than one number and have two or more variable quantities).
        (2) *Supervised Learning* refers to problems where the data has attributes which we want to predict; this can be either:
            (a) *Classification*, where samples belong to certain classes and we learn from labeled data how to predict the classes of unlabeled data
            (b) Or *regression*, where the prediction (the output) is a continuous (can take on any value within a range) variable
        (3) Scikit-Learn has a number of datasets that can be easily imported (like normal python imports)
        (4) The *.data* member contains the features of the samples, while the *.target* member gives the number corresponding to the *image* (correct version) which we are trying to learn.
        (5) An *estimator* for classification is a python object that implements the methods *fit(X, y)*, which is used to train the model by 'fitting' it to the data, and *predict(T)*, which .

B. First Image Classification with Scikit-Learn:
1. In preparation for my usage of Scikit-learn to predict if collected data indicates the presence of an exoplanet, I need to learn how to classify images in a deep learning setting. Once I understand this, I will be able to begin classifying various light curves that I produce using the code discussed above within a Jupyter notebook.
2. For my first attempt at image classification, I followed this documentation and example, where I produce a model to classify fairly poor quality images of numbers. It is my plan to then apply the knowledge I gain in the following attempt to light curves in the manner mentioned previously.
3. After navigating a small import error (and realizing that I simply needed to enter python and leave the command prompt default function), I was able to begin working on the digit image recognition model.

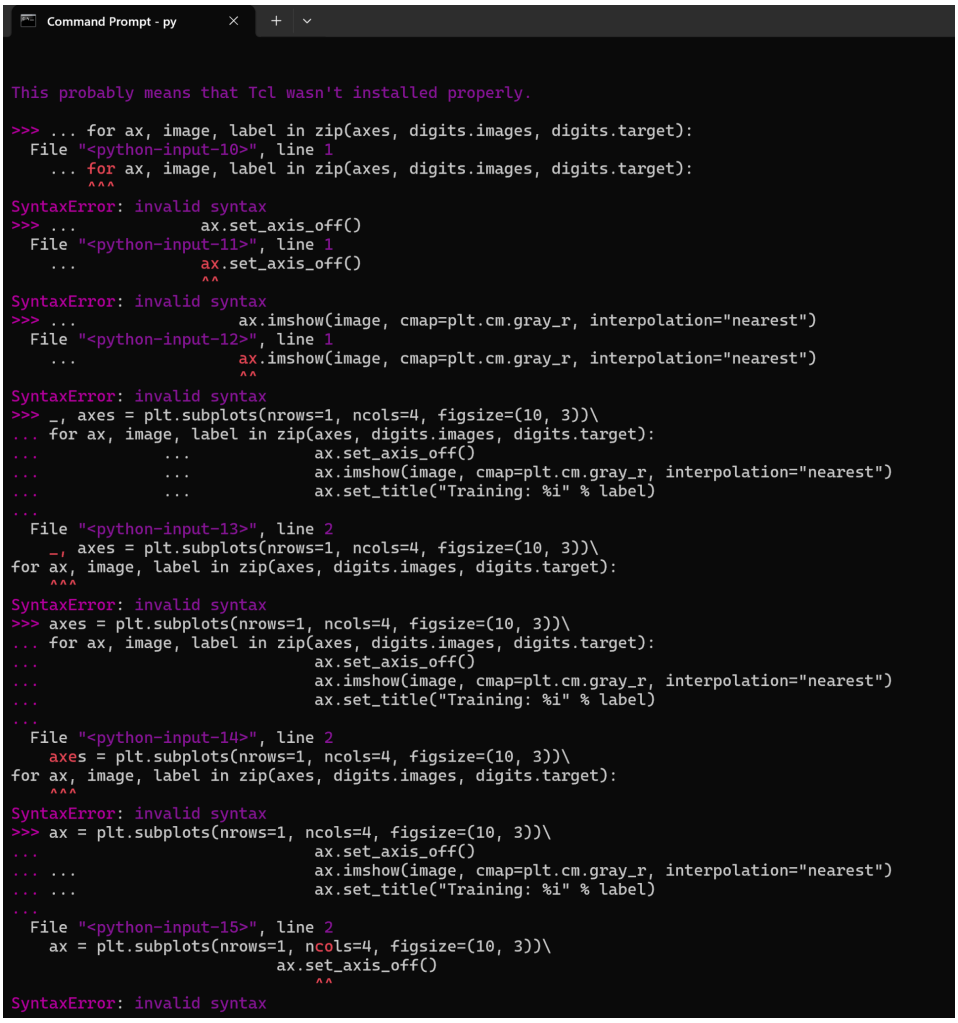


Figure IV.3 - As seen here, I made several attempts at producing images from the initial dataset, but encountered a number of syntax errors throughout the code.

a) First, I encountered a number of significant syntax errors (see figure IV.3) when attempting to produce the images contained within the Scikit-Learn database. In order to resolve these errors, I spoke with Mr. Adu and began using the linked matplotlib documentation to check every part of this initial code.
    (1) With this research, I recognized that there was a blank before the term ‘axes’ in the sample code, a blank which is filled in the matplotlib examples (see figure IV.4). Thus, I made a second attempt at running the code, this time with that blank filled in with the word ‘fig.’

```
# using the variable ax for single a Axes
fig, ax = plt.subplots()

# using the variable axs for multiple Axes
fig, axs = plt.subplots(2, 2)

# using tuple unpacking for multiple Axes
fig, (ax1, ax2) = plt.subplots(1, 2)
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2)
```

Figure IV.4 - Matplotlib documentation examples for the plt.subplots function, which I use in the first portion of my code to visualize the data (prior to assembling the model). Note the presence of 'fig' prior to the term 'axs.'

b) That second attempt failed with essentially no change in the quantity and depth of errors I received. I concluded that something significant must be incorrect with my application, and so I conducted some additional research on the subject.
    (1) In that research, I also learned more about the crux of the code that allows the algorithm to conduct its classification techniques: the **svm.SVC() method**.
        (a) This is called the *support vector classifier*, which establishes a kind of line of best fit in the center of a plot (somewhat akin to that of the center red line on the light curves above)

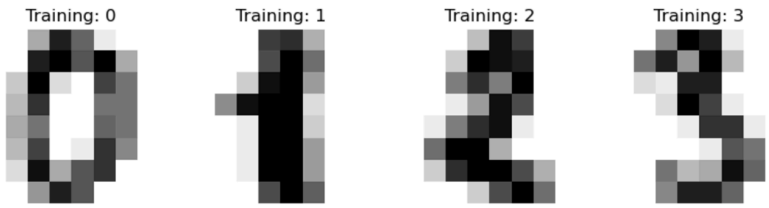
c) Then I realized that rather than writing the example machine learning code in the command prompt of my system, I needed to write it in a Jupyter notebook. Once I did so, the results were much more promising, and the sample code functioned very well. In addition, I was able to make slight modifications to the sample code in order to increase both my understanding of the code itself and of the workings of deep learning overall.

jupyter SRPII.MachineLearningTestA Last Checkpoint: 3 minutes ago

File Edit View Run Kernel Settings Help Trusted

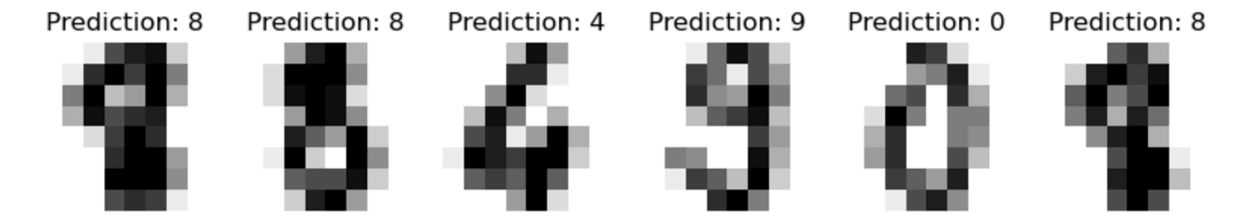
Code JupyterLab Python 3 (ipykernel)

[1]:
```
import matplotlib.pyplot as plt

from sklearn import datasets, metrics, svm
from sklearn.model_selection import train_test_split

digits = datasets.load_digits()

_, axes = plt.subplots(nrows=1, ncols=4, figsize=(10, 3))
for ax, image, label in zip(axes, digits.images, digits.target):
    ax.set_axis_off()
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation="nearest")
    ax.set_title("Training: %i" % label)
```



[ ]:

```
import matplotlib.pyplot as plt

from sklearn import datasets, metrics, svm
from sklearn.model_selection import train_test_split

digits = datasets.load_digits()

_, axes = plt.subplots(nrows=1, ncols=4, figsize=(10, 3))
for ax, image, label in zip(axes, digits.images, digits.target):
    ax.set_axis_off()
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation="nearest")
    ax.set_title("Training: %i" % label)

# flatten the images
n_samples = len(digits.images)
data = digits.images.reshape((n_samples, -1))

# Create a classifier: a support vector classifier
clf = svm.SVC(gamma=0.001)

# Split data into 50% train and 50% test subsets
X_train, X_test, y_train, y_test = train_test_split(
    data, digits.target, test_size=0.5, shuffle=False
)

# Learn the digits on the train subset
clf.fit(X_train, y_train)

# Predict the value of the digit on the test subset
predicted = clf.predict(X_test)

_, axes = plt.subplots(nrows=1, ncols=6, figsize=(10, 3))
for ax, image, prediction in zip(axes, X_test, predicted):
    ax.set_axis_off()
    image = image.reshape(8, 8)
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation="nearest")
    ax.set_title(f"Prediction: {prediction}")
```




```
print(
    f"Classification report for classifier {clf}:\n"
    f"{metrics.classification_report(y_test, predicted)}\n")
```

```
Classification report for classifier SVC(gamma=0.001):
              precision    recall  f1-score   support

           0       1.00      0.99      0.99        88
           1       0.99      0.97      0.98        91
           2       0.99      0.99      0.99        86
           3       0.98      0.87      0.92        91
           4       0.99      0.96      0.97        92
           5       0.95      0.97      0.96        91
           6       0.99      0.99      0.99        91
           7       0.96      0.99      0.97        89
           8       0.94      1.00      0.97        88
           9       0.93      0.98      0.95        92

    accuracy                           0.97       899
   macro avg       0.97      0.97      0.97       899
weighted avg       0.97      0.97      0.97       899
```

```
[ ]:
```

Now, I will have to build a dataset of both confirmed exoplanet light curves and light curves that have been confirmed as non-exoplanets. Then, I can modify the algorithm above in order to give it the ability to sort light curves I feed it into those of confirmed exoplanets and those of non-exoplanetary objects.

However, due to the nature of the NASA internship being withdrawn (likely due to financial or time constraints), I believe it is best to continue this project on my own and focus my SRP project more on any research that Professor Marka may be able to offer me an opportunity to assist in.

V. 12.3.24-12.16.24 Entry: Assembling Light Curve Dataset & Connection with Prof. Marka:

A. Using MAST to assemble a Confirmed Exoplanet Database:

1. In order to prepare a database for the deep learning algorithm, I began processing a number of different light curves from the MAST database using the methodology I proved earlier.

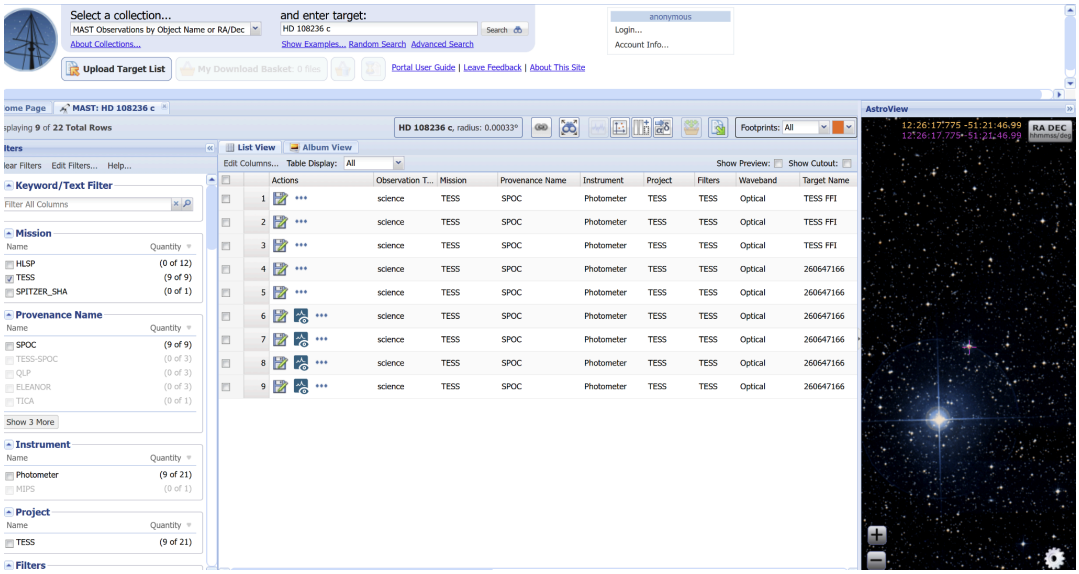
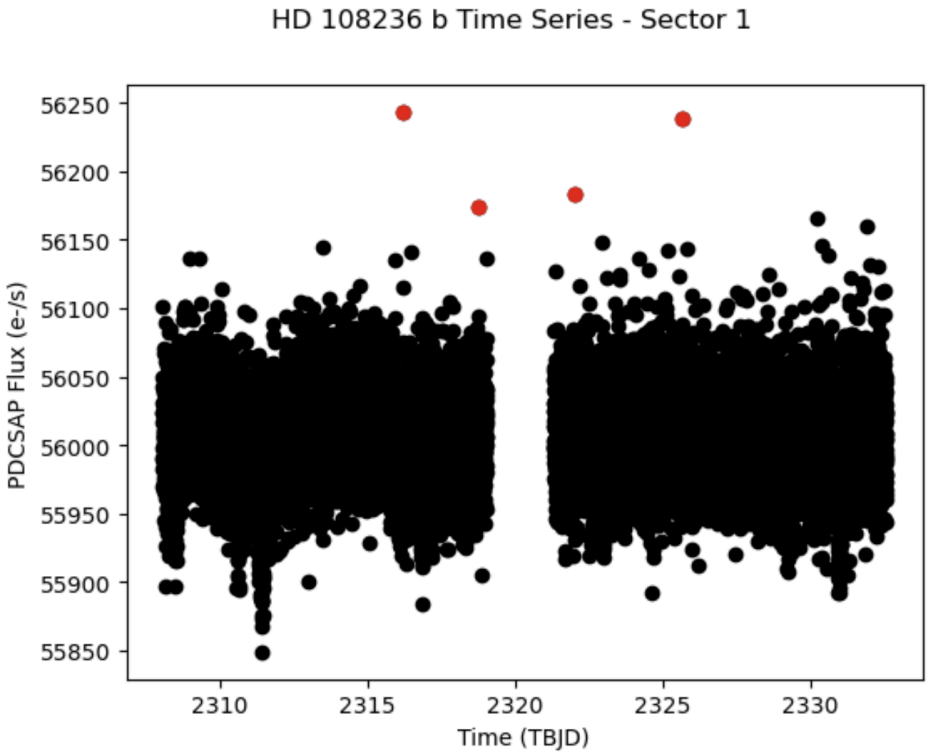
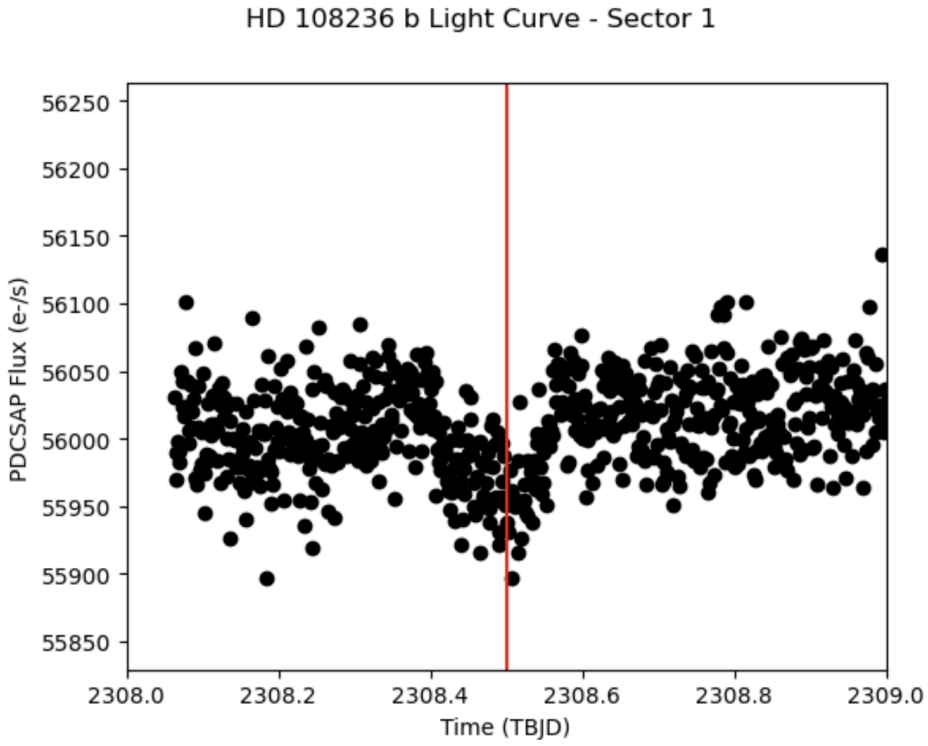


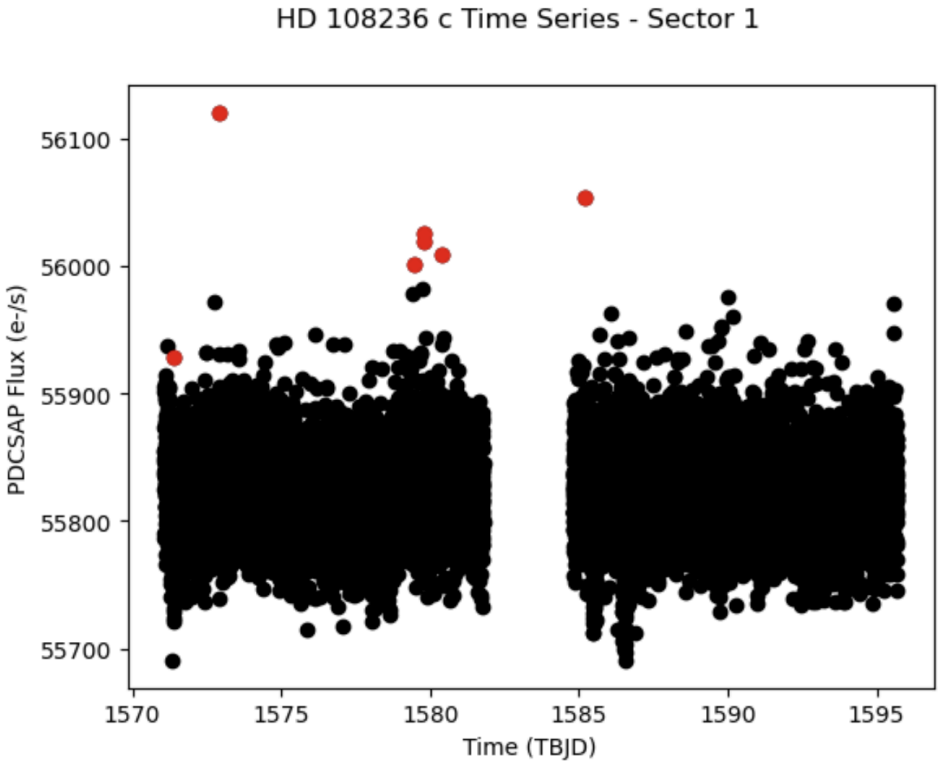
Figure V.1 - Sample MAST data portal search results, in this case, for the confirmed exoplanet HD 108236 c

2. After cross-checking the exoplanet database in MAST with the NASA JPL Exoplanet Archive, I was able to determine which light curves were of confirmed exoplanets. Thus, With a combination of Jupyter Notebook via Anaconda, I developed the following time series and light curves:
    a) HD 108236 b:




HD 108236 b Light Curve - Sector 1
PDCSAP Flux (e-/s)
56250
56200
56150
56100
56050
56000
55950
55900
55850
2308.0
2308.2
2308.4
2308.6
2308.8
2309.0
Time (TBJD)

b) HD 108236 c


HD 108236 c Time Series - Sector 1
PDCSAP Flux (e-/s)
56100
56000
55900
55800
55700
1570
1575
1580
1585
1590
1595
Time (TBJD)

c) I completed other light curves, which I intend to add to this document later.
d) In order to import this into the deep learning algorithm,

B. Email to Professor Marka:

1. In addition to beginning the datasets, I contacted Professor Marka about what project he would like me to work on in the New Year with the following email:
2. Dear Professor Marka,
3. Thank you again for your letter of recommendation and for your allowing me to continue researching with you this year. I simply wanted to reach out to hear about any potential research project ideas you have for me to pursue going forward. Whenever it is convenient for you, simply let me know, and we can set up a meeting in which we can go over those potential ideas.
4. Thanks, and enjoy your weekend.
5. Sincerely,
6. Brian LaMons

[See next page]

VI. 12.30.24-1.10.25 Entry: Unconfirmed Exoplanets, Further Research on Building Deep Learning Algorithms, & Beginning Modifications:

A. Establishing an Unconfirmed Exoplanet Dataset:

1. In prior meetings, the concern was introduced that I would be unable to produce a dataset of unconfirmed exoplanet light curves. However, by sifting through publications of studies online, I realized that, when missions like TESS detect a transit and produce a light curve of a planet that is not confirmed, it is marked as a TOI (*TESS Object of Interest*).
    a) Subsequently, I searched for TOIs such as TOI-101.01 or TOI-1251.01, both of which scientific journals have verified to be unconfirmed exoplanets, in the MAST Database. Thus far, MAST has been reliable, although in this scenario, it did not produce any results for either of those TOIs. This led me to believe that MAST only contained data on confirmed exoplanets.

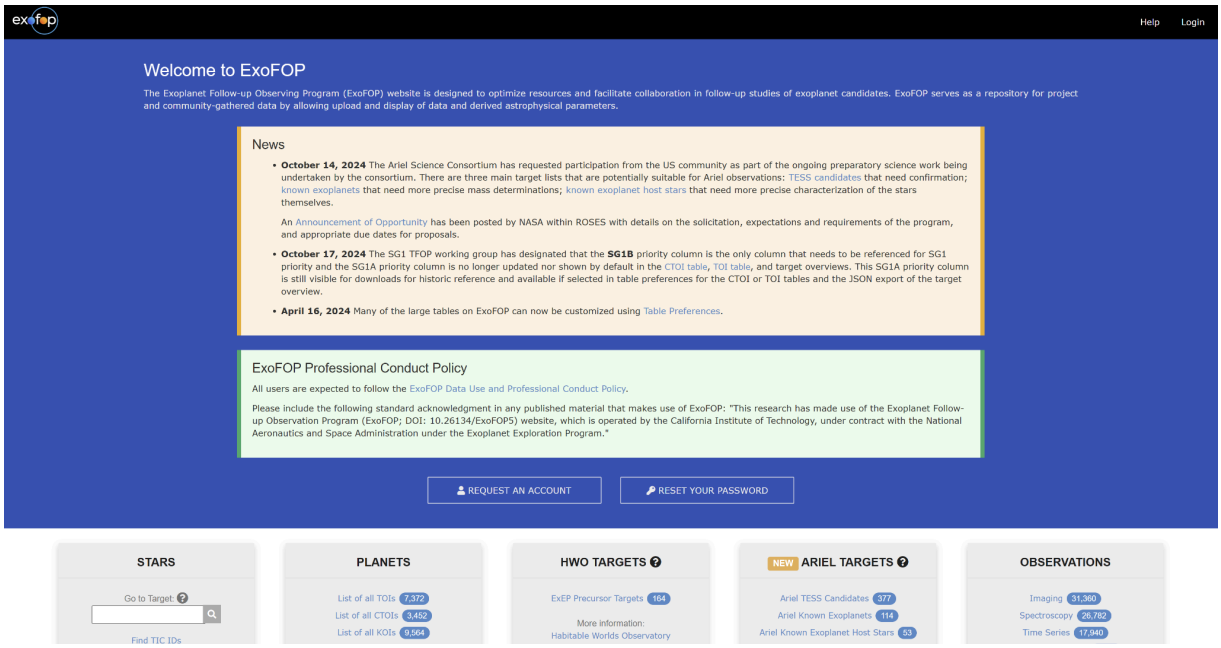
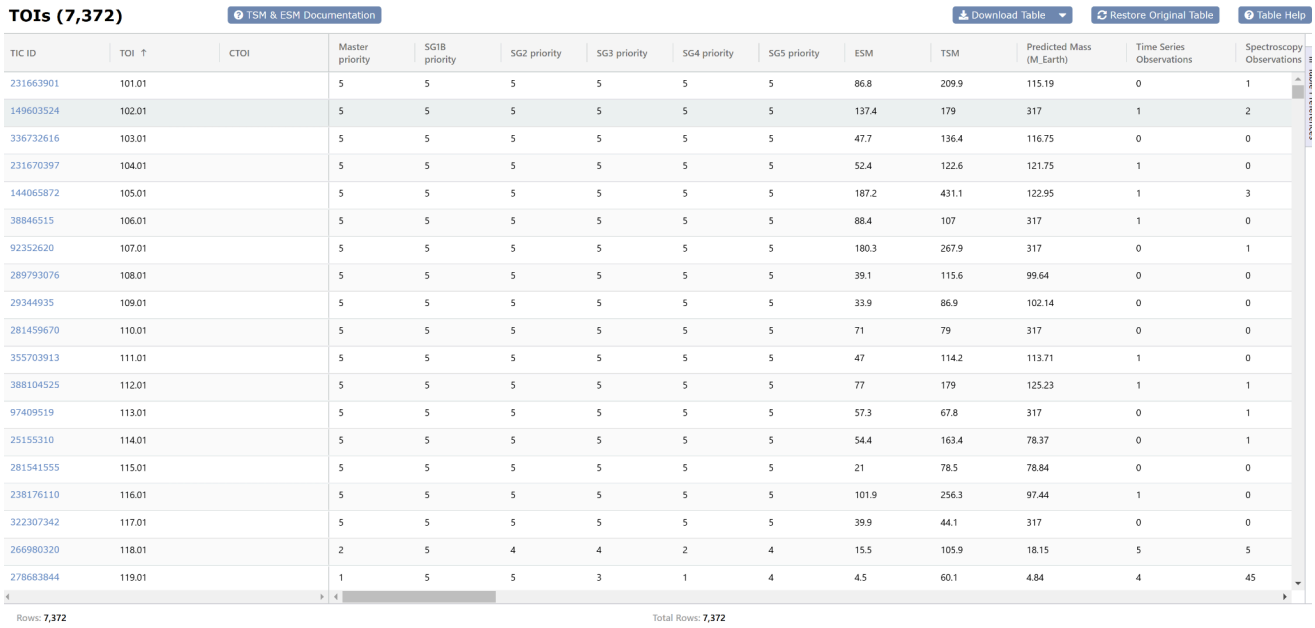


Figure VI.1 - Home page of the ExoFOP *(Exoplanet Follow-up Observing Program) Database, which is a repository of transit data (including light curves) for objects like TOIs.*

    b) As a result, I continued my research elsewhere until I found the ExoFOP Database, pictured above. This database, maintained by Caltech, is specifically designed to support further exoplanet research by providing access to the transit data of objects that have <u>not</u> been confirmed as exoplanets.
        (1) **TOI ExoFOP Link (includes over 7,000 TOIs, see below): https://exofop.ipac.caltech.edu/tess/view_toi.php**



TOIs (7,372) | TSM & ESM Documentation | Download Table | Restore Original Table | Table Help

TIC ID	TOI ↑	CTOI	Master priority	SG1B priority	SG2 priority	SG3 priority	SG4 priority	SG5 priority	ESM	TSM	Predicted Mass (M_Earth)	Time Series Observations	Spectroscopy Observations
231663901	101.01		5	5	5	5	5	5	86.8	209.9	115.19	0	1
149603524	102.01		5	5	5	5	5	5	137.4	179	317	1	2
336732616	103.01		5	5	5	5	5	5	47.7	136.4	116.75	0	0
231670397	104.01		5	5	5	5	5	5	52.4	122.6	121.75	1	0
144065872	105.01		5	5	5	5	5	5	187.2	431.1	122.95	1	3
38846515	106.01		5	5	5	5	5	5	88.4	107	317	1	0
92352620	107.01		5	5	5	5	5	5	180.3	267.9	317	0	1
289793076	108.01		5	5	5	5	5	5	39.1	115.6	99.64	0	0
29344935	109.01		5	5	5	5	5	5	33.9	86.9	102.14	0	0
281459670	110.01		5	5	5	5	5	5	71	79	317	0	0
355703913	111.01		5	5	5	5	5	5	47	114.2	113.71	1	0
388104525	112.01		5	5	5	5	5	5	77	179	125.23	1	1
97409519	113.01		5	5	5	5	5	5	57.3	67.8	317	0	1
25155310	114.01		5	5	5	5	5	5	54.4	163.4	78.37	0	1
281541555	115.01		5	5	5	5	5	5	21	78.5	78.84	0	0
238176110	116.01		5	5	5	5	5	5	101.9	256.3	97.44	1	0
322307342	117.01		5	5	5	5	5	5	39.9	44.1	317	0	0
266980320	118.01		2	5	4	4	2	4	15.5	105.9	18.15	5	5
278683844	119.01		1	5	5	3	1	4	4.5	60.1	4.84	4	45

Rows: 7,372 Total Rows: 7,372

Figure VI.2 - TOI list on the ExoFOP Database. As seen, there are plenty of unconfirmed exoplanets with which to create a dataset of unconfirmed light curves.

c) Then, I decided to search ExoFOP for observational data on TOI-101.01, and I was redirected to the page shown below. From there, I was able to download several different files containing a wide variety of data on TOI-101.01. As it so happened, one of them included the exact graphs I needed: the time series and light curve, specifically those of relative flux vs. time (or phase, in some cases).

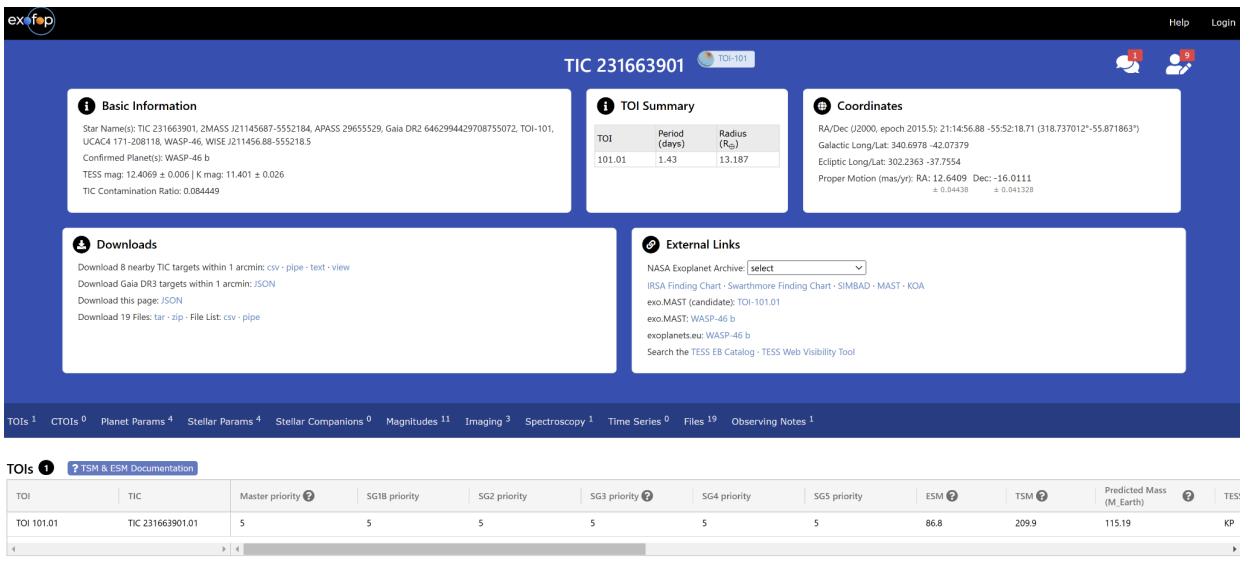


Figure VI.3 - ExoFOP Database page for the TOI-101 system, in which TOI-101.01 resides.

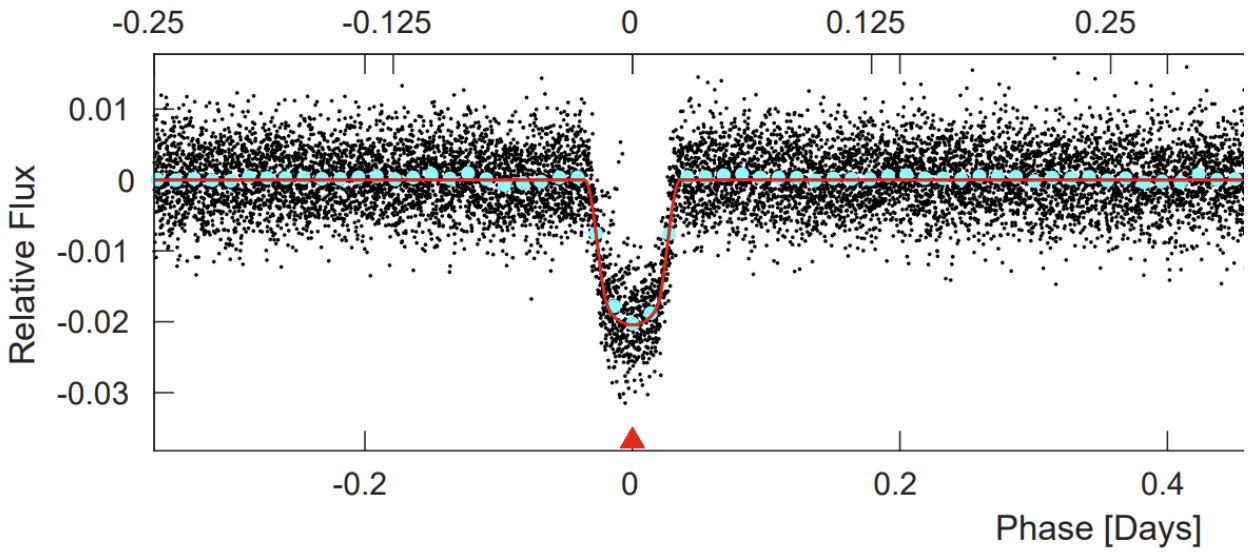


Figure VI.4 - Light curve for the unconfirmed exoplanet TOI-101.01, where time is in phases rather than TBJD. However, this ultimately does not make a significant difference, as both are in units of days.

d) NOTE: ExoFOP has links to the NASA Exoplanet Archive (maintained by JPL), the initial database I attempted to use to produce light curves, in the event that I require the services of one database to verify the graphs of the other.

2. With this newfound access, I was able to produce the following dataset of unconfirmed exoplanet light curves that I can then use for my algorithm:

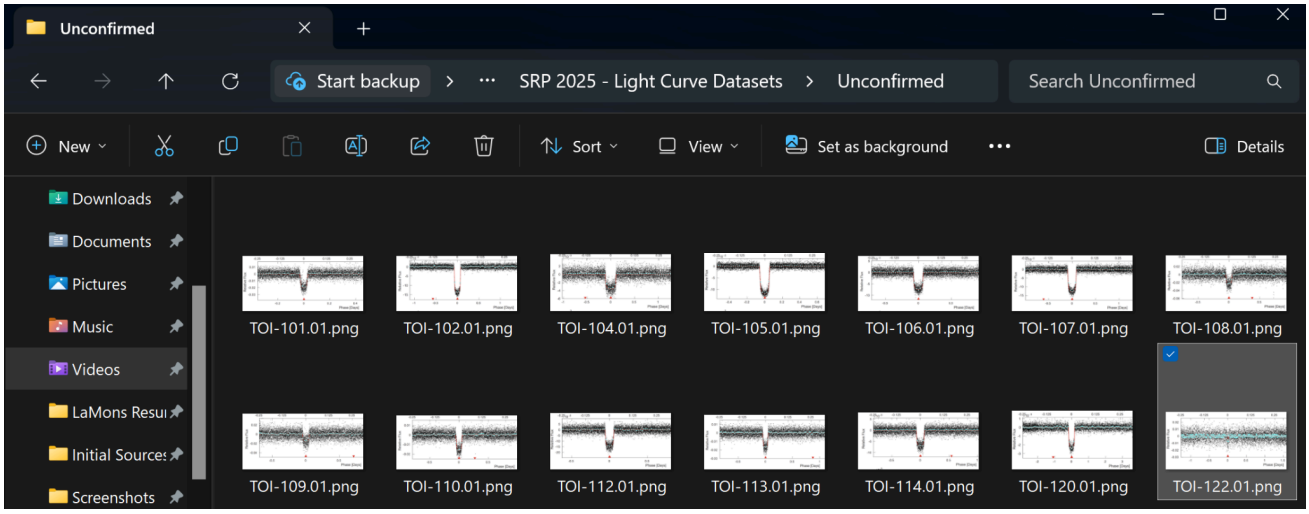
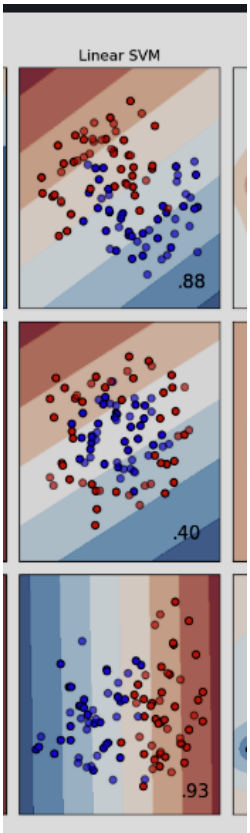


Figure VI.5 - Screenshot of the "Unconfirmed" folder of my dataset, which now contains more than 14 light curves as derived from documentation on the ExoFOP Database.

B. Research on and First Assembly of the Deep Learning Algorithm:

1. The first component necessary is preparing data, which I am able to do via sorting light curves I produce using the above code and MAST or the ExoFOP search engine into two categories: light curves of confirmed exoplanets and light curves of non-confirmed exoplanets.
    a) In order to access the data stored on my computer's hard drive, I have to use the *import os* command. That way, I can actually utilize the datasets shown to train the algorithm (and later, test it).

2. Once the data is sorted and prepared, one can begin to build the image classifier in Scikit-Learn, importing and reformatting the input data (in this case, the images) in preparation for analyzing the data via the deep learning algorithm.
    a) Depending on what classifier you are using, you will have to resize and format the input images differently.
    b) Based on Scikit-Learn documentation (seen below), I believe that it might be best to try a Linear SVM classifier, as it will deal with the vertical rectangle of each graph in which the 'curve' portion of the light curve occurs.



Figures VI.6.1-2 - Scikit-Learn website analysis of different classifiers. It specifically mentions and shows in the pictogram to the right, how linear SVMs are well-adapted to high-dimensional spaces. Light curves, as they have numerous data points on a graph, would be considered "high-dimensional." Plus, the curve of a light curve fits directly into the vertical slice depicted in the bottom image of the pictogram.

c) However, for this first trial, I am going to start with a basic classifier simply to test how it functions and how the algorithm handles the light curve datasets. The issue of which classifier to use will likely be a topic of further consideration in the future.

d) As a result, for this first component of initial image processing, I produced the following code (in Jupyter Notebook with Scikit-Learn):

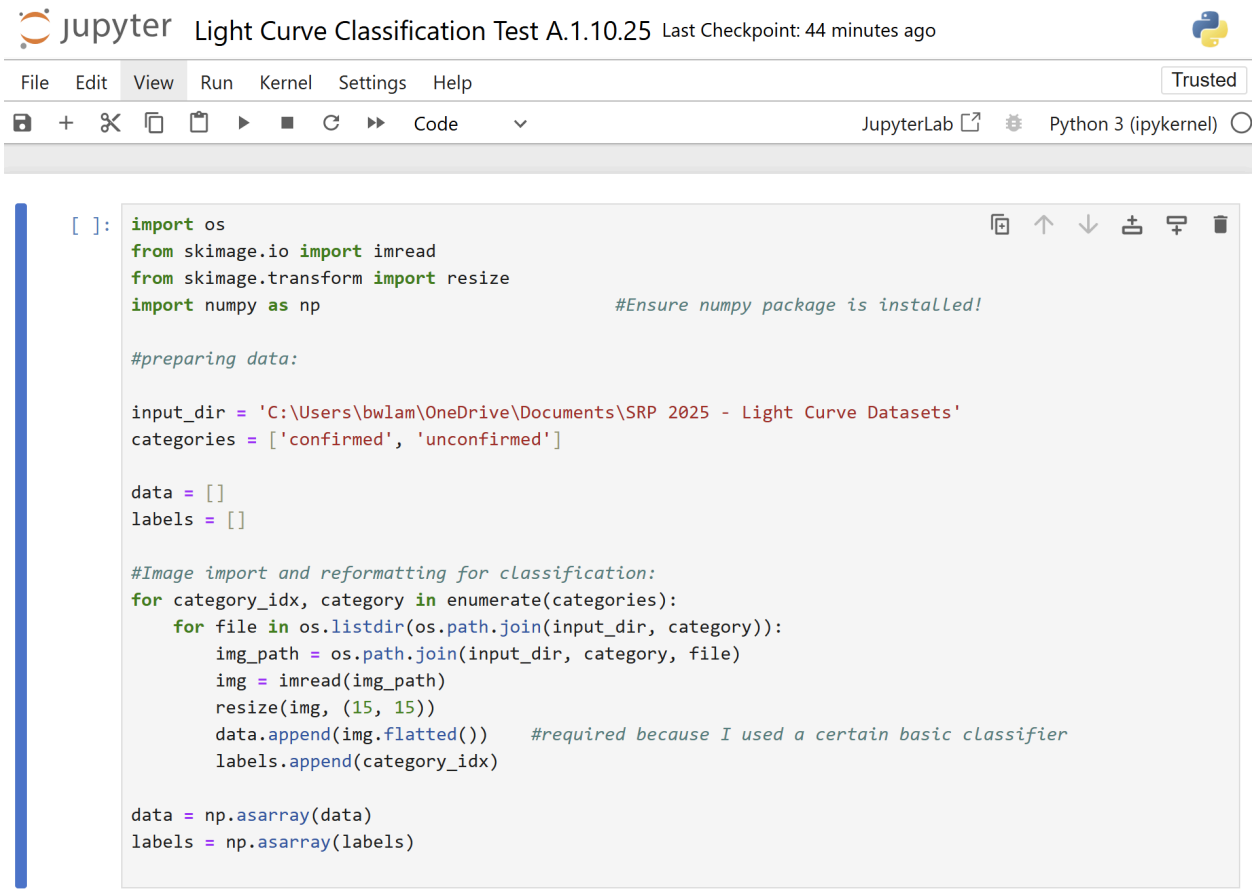


Figure VI.7 - Jupyter Notebook file in which I have coded the initial preparation stages for the datasets. However, it has an additional method (flatted()), on account of the simple classifier I will use later in the code.

e) NOTE: My research also indicated that, perhaps, it might be best to try using deep learning in **Keras with TensorFlow** (the other tool NASA utilizes for light curve classification). This is due to the fact that the capabilities of Keras might involve classifiers more suited to light curves than those available in Scikit-Learn. That possibility is most certainly worthy of further consideration.

3. In order to actually support this first portion of the code, you will notice that the package *numpy* is required. Thus, I had to install numpy through the command prompt. Although doing so was not a lengthy process, it did delay some further development of the algorithm.

4. Next, I had to split the data into a training set and a test set in order to begin training the deep learning model.
    a) To do that, I replicated some code fragments from the first deep learning algorithm I designed (see figures IV.5-6). These included the following:

```
# Split data into 50% train and 50% test subsets
X_train, X_test, y_train, y_test = train_test_split(
    data, digits.target, test_size=0.5, shuffle=False
)

# Learn the digits on the train subset
clf.fit(X_train, y_train)
```

Figure VI.8 - Scikit-Learn code for splitting datasets into training and testing (specifically, a 50% split). Yet, the second portion of the above code will require a classifier to execute, which I will add to the code for the next cycle.

    b) I added this to my new algorithm so that now I am prepared to begin the classifier portion of the code.

C. Plan for Next Cycle:
    1. For the next cycle, I plan to continue the implementation of the research I conducted and took notes on above (section VI-B) by completing the classifier in the code (I will also conduct further research on

    2. Then, I will run preliminary tests of this first iteration of my deep learning algorithm. In doing so, I will not only test the algorithm itself, but I will also test out my datasets for the first time (including the new unconfirmed exoplanet one that I just completed over the break).

    3. In addition, I think it might be a good idea to send another email to Professor Marka to follow-up on my request for further research, so long as Dr. Matone deems that to be appropriate.

4. Note: switch focus to taking in a light curve and processing it to output data such as the orbital period and distance to its host star.

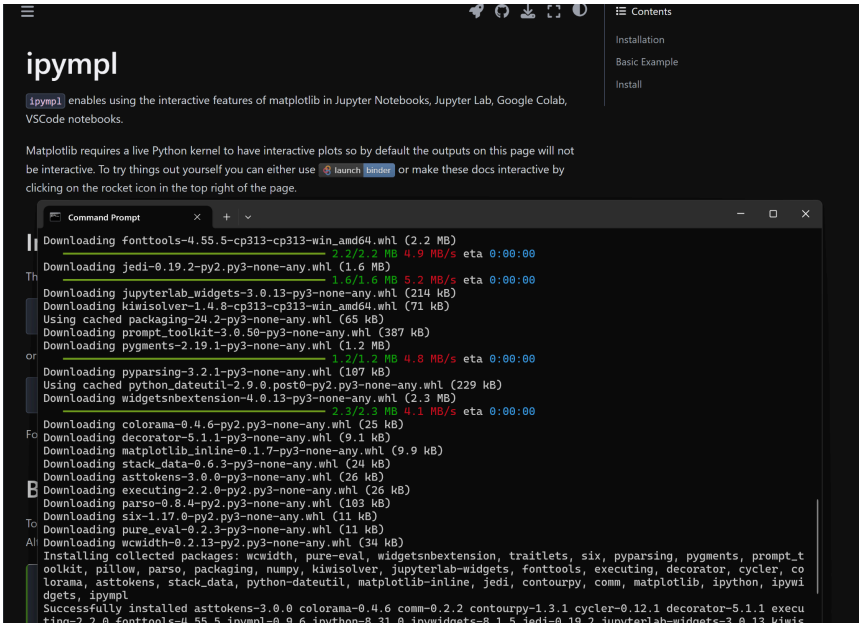
VII. 1.15-1.24.25 Entry: Completing the Light Curve Processing Algorithm:

A. General Parameters:

1. The algorithm should take a light curve as input, and then find the transit time by measuring the x axis distance between one end of the dip in the light curve to another.
2. In addition, by measuring the x axis distance in between different curves on an inputted light curve time series, it will find the orbital period (P).
3. Then, it should calculate, using Kepler's Third Law, the semi-major axis (a) of the exoplanet, or the distance between the planet and its host star.

AIMS:

1) Produce light curve and time series from database & display both
2) Take in coordinates for the distance between the sides of a curve dip and the distance between curves in a time series
    a) By clicking points on a separate, new plot below to mirror those parts of the light curve and time series, I can find the coordinates of the points that correspond to those parts
        i) In order to accomplish this (as I am continuing to operate within a Jupyter notebook), after some research, I realized that I needed to download the ipympl package. This library is specifically used for activating interactive features in Jupyter notebook such as clicking on a graph as I aim to do here.



        ii) However, I continued to encounter a multitude of errors (especially with the interactive portion) as I worked through producing the algorithm.

    b) With the coordinate data, I can calculate the distances as functions in Python, and in doing so determine and display the transit period and the orbital period.

3) Then, by creating additional methods, I can have the program calculate the semi-major axis using Kepler's Third Law (Where P is the distance between time series points, the third [p3] and fourth [p4] collected coordinates, and a = $\sqrt[3]{P^2} = \sqrt[3]{(p4_x - p3_x)^2}$
    a) In addition to finding the semi-major axis, by combining the Third Law with Newton's law of gravity, I can have the algorithm calculate the mass of the star the given planet orbits ($M = \frac{4\pi^2}{GT^2}a^3$, where T = P = orbital period).
    b) In order to accomplish this, I had to import the additional package *SciPy*, so that I might be able to access accurate numbers for physical constants like G.

The Algorithm Code thus far.

```
#Component II: Mimic-Plot Point Selection & Data Collection:

fig, ax = mp.subplots()
ax.plot()
ax.set_title('Sample Data Collection Plot: Enter Coordinates:')

xcoords = []
def onclick(event):
    xdata, ydata = event.xdata, event.ydata
    xcoords.append(xdata)
    print(f' X Coordinates {xdata:.2f}')

cid = fig.canvas.mpl_connect('button_press_event', onclick)
mp.show()

#Component III: Calculation Methods

t_final = xcoords[4]
t_initial = xcoords[3]

def semi_major_calc(t_final, t_initial):
    a = ((t_final - t_initial)(t_final - t_initial))**(1/3)
    return a

def stellar_mass_calc(a, t_final, t_initial):
    m = (((4*scipy.constants.pi)**(2))/(((a**(2))*scipy.constants.G)))
    return m

a = semi_major_calc(t_final, t_initial)
m = stellar_mass_calc(a, t_final, t_initial)
print("Semi-Major Axis:" + a)
print("Mass of Orbiting Star:" + m)
```

4) Perhaps also include a percent error calculation, where the program compares its values with the generally accepted values found in the NASA Exoplanet Archive.
5) Therefore, the algorithm Should display the following:
    a) Light curve for the given planet
    b) Time series for the given planet
    c) Graphing interface to plot approximate points
    d) Transit period and corresponding percent error
    e) Orbital period and corresponding percent error
    f) Semi-major axis and corresponding percent error
    g) Approximate mass of central star and corresponding percent error (if applicable)

VIII. 2.5-2.10.25 Entry: Journal Research, Fixing Issues with Processing Algorithm, & Continuing the Light Curve Sort Algorithm:

A. Research on Documentation about Machine Learning & Exoplanet Identification / Light Curve Sorting:
   1. https://pmc.ncbi.nlm.nih.gov/articles/PMC9132280/#sec003
      a) "The classification task consists of sorting every light curve into two categories: Planet candidate and False positive. Different kinds of deep learning approaches have been used for this application. In our approach, a convolutional neural network was chosen since they outperform artificial neural networks in classification tasks where the data is spatially aligned such as image or audio [30]. It is because CNN leverages the spatial structure of the output detecting local features which only need to be learned once, therefore the number of trainable parameters, the memory usage, and the number of computations of the desired output will decrease."
   2. https://arxiv.org/abs/2301.01371
      a) This article demonstrates how "deep learning techniques such as neural networks have proved effective at differentiating promising astrophysical eclipsing candidates from other phenomena such as stellar variability and systematic instrumental effects in an efficient, unbiased and sustainable manner."
      b) In terms of solutions, they describe using convolutional neural networks in order to

B. Fixing Errors in the Light Curve Processing Algorithm:
   1. Time and again, I received the "TypeError" error message, indicating that, for some reason, the decimal value I was using (in this case, the initial and final times for the light curve), was not callable.
   2. Thus, I began editing the code, searching for a solution to this error, as well as conducting some research on the subject so that I would be able to understand and solve the problem faster.

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[6], line 90
     87 def stellar_mass_calc(a, t_final, t_initial):
     88     return (((4*scipy.constants.pi)**(2))/(((a**(2))*scipy.constants.G)))
---> 90 print("Semi-Major Axis:" + semi_major_calc(t_final, t_initial))
     91 print("Mass of Orbiting Star:" + stellar_mass_calc(a, t_final, t_initial))

Cell In[6], line 85, in semi_major_calc(t_final, t_initial)
     84 def semi_major_calc(t_final, t_initial):
---> 85     return ((t_final - t_initial)(t_final - t_initial))**(1/3)

TypeError: 'float' object is not callable
```

   3. After some research, I realized that I was likely missing a multiplication operator or simply had too many parentheses within my function

definitions, and so I debugged my code and fixed the error. After a few minor error revisions, the code was fixed (see below):

```
def semi_major_calc(tf, ti):
    difference = (tf - ti) * (tf - ti)
    return difference**(1/3)

semi_major_axis = semi_major_calc(t_final, t_initial)

def stellar_mass_calc(a):
    return ((4*scipy.constants.pi)**(2))/(((a**(2))*scipy.constants.G))

print("Semi-Major Axis: " + str(semi_major_axis))
print("Mass of Orbiting Star: " + str(stellar_mass_calc(semi_major_axis)))
```

C. Continuing Progress on Deep Learning Sort Algorithm:
   1. I began researching suggestions and referencing current documentation on how to split a data set into training and testing, as I am currently in the second phase of writing the light curve sort algorithm.
      a) As a reminder, there are 4 phases or components to my deep learning algorithm:
         (1) Import and extract data from hard drive, as well as flatten and prepare it for classification (completed previously during the initial construction of the algorithm)
         (2) Split data into a training set and a test set (what I am working on now)
         (3) Implement and run the classifier code, the heart of the algorithm
         (4) Report and analyze results of the classifier to verify accuracy and find any problems that must be solved or edits that need to be made.
   2. So, in order to complete the split code and phase 2, I implemented the following:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.2,
    shuffle=True,
    random_state=42,
)
```

a) This ensures that the test and train datasets are roughly similar (in this case, that they are composed of approximately equal amounts of confirmed and unconfirmed light curves). In addition, in my research, I found a way to verify that similarity between the two data sets, which is seen below:

```
def plot_bar(y, loc='left', relative=True):
    width = 0.35
    if loc == 'left':
        n = -0.5
    elif loc == 'right':
        n = 0.5

    # calculate counts per type and sort, to ensure their order
    unique, counts = np.unique(y, return_counts=True)
    sorted_index = np.argsort(unique)
    unique = unique[sorted_index]

    if relative:
        # plot as a percentage
        counts = 100*counts[sorted_index]/len(y)
        ylabel_text = '% count'
    else:
        # plot counts
        counts = counts[sorted_index]
        ylabel_text = 'count'

    xtemp = np.arange(len(unique))

    plt.bar(xtemp + n*width, counts, align='center', alpha=.7, width=width)
    plt.xticks(xtemp, unique, rotation=45)
    plt.xlabel('equipment type')
    plt.ylabel(ylabel_text)

plt.suptitle('relative amount of photos per type')
plot_bar(y_train, loc='left')
plot_bar(y_test, loc='right')
plt.legend([
    'train ({0} photos)'.format(len(y_train)),
    'test ({0} photos)'.format(len(y_test))
]);
```

3. Now that I have largely completed phase two of the algorithm, going forward, I will look to complete phase three (and perhaps four) in order to begin testing the algorithm for the first time. As break is fast approaching, I plan to work on the algorithm for a significant amount of time during the President's Day Week so that I can return with those above goals completed and a functioning preliminary algorithm.

IX. 2.18-2.24.25 Entry: Emailing MIT Investigators and Researching / Coding Phases 3 & 4 of Light Curve Sort Algorithm:

A. Email sent to MIT researchers with regard to their work (see journal article summarized above - Identifying Exoplanets with Deep Learning. V. Improved Light Curve Classification for TESS Full Frame Image Observations):

1. "Dear Mr. Tey,

2. I'm Brian LaMons, a junior at Regis High School in New York City. I recently read your paper Identifying Exoplanets with Deep Learning. V. Improved Light Curve Classification for TESS Full Frame Image Observations from 2023 and found it to be very interesting. In emailing you, I was hoping to learn more about the specifics of your methodology and how you were able to produce an effective classifier with deep learning.

3. I want to learn more about how you were able to do this not only because I am fascinated by exoplanets and our ongoing quest to identify and discover more of them, but also because it relates to a project I am working on as part of Regis' Science Research Project (SRP) class. I decided to create my own deep learning algorithm to classify TESS light curves as either likely to be of confirmed exoplanets or likely to be of other astrophysical phenomena / celestial bodies. So far, I have had success with extracting light curves from the NASA Exoplanet Archive and MAST (Mikulski Archive for Space Telescopes) and have been able to write a significant portion of my deep learning algorithm using Scikit-Learn and Jupyter notebook.

4. After reading your paper, I would like to discuss your own experience in producing a similar algorithm, for I am looking for some guidance from experts such as yourself as I continue to work on my algorithm. If possible, I would like to discuss with you about your methods and experience in producing a light curve classifying algorithm, as well as any recommendations you might have for me as I proceed with my project.

5. I am very interested in this subject and your work, so I appreciate any time or information you can provide.

6. Your website said that you like receiving random emails, so I hope you enjoyed this one. Thank you and have a great night."

7. Unfortunately, the email address that I had of the principal investigator, Evan Tey, no longer exists, as I received a "return to sender" message multiple times. As a result, I emailed a secondary investigator, an MIT professor, in the hopes that he would read my email and respond to it.

B. Working on Phases 3 & 4 of the Light Curve Sort Algorithm:
   1. Phase **three** deals with the core of the algorithm, the image processing and classifier.
      a) In order to determine which kind of classifier I should use, I searched a number of data science-related sites, and found two possible options:
         (1) The HOG-SVM & SGD (more traditional and efficient; however, it is usually used on full images, not simply graphs as in the case of my datasets)
            (a) See (*processing* section): Scikit-Learn Image Classifier Sample & Outline

         (2) A GNN (graph neural network; more experimental and complicated; however, it is designed specifically to process data structured in a graph, like those in my datasets)
            (a) See: Intro & Sample of Graph Neural Networks

      b) Due to the fact that Scikit-Learn, the deep learning library I have been using for the light curve sort algorithm thus far, is more tailored to the HOG-SVM and SGD method, I decided to go with that option.
         (1) However, I believe that it would be worth my time (in the near future) to attempt to produce a similar light curve sort algorithm with a different deep learning library, PyTorch, and a GNN as a classifier (GNNs are more suited to implementation in PyTorch). Then, I can compare the accuracy of each algorithm to see which processing method and classifier is more effective.

c) I thus implemented the HOG-SVM processing algorithm as follows:

```
#Phase III: Processing & Classifier:

confirmcurve = imread('wasp126b.png', as_gray=True)

# scale down the image to one third
confirmcurve = rescale(confirmcurve, 1/3, mode='reflect')
# calculate the hog and return a visual representation.
confirmcurve_hog, confirmcurve_hog_img = hog(
    confirmcurve, pixels_per_cell=(14,14),
    cells_per_block=(2, 2),
    orientations=9,
    visualize=True,
    block_norm='L2-Hys')

fig, ax = plt.subplots(1,2)
fig.set_size_inches(8,6)
# remove ticks and their labels
[a.tick_params(bottom=False, left=False, labelbottom=False, labelleft=False)
    for a in ax]

#Showing
ax[0].imshow(confirmcurve, cmap='gray')
ax[0].set_title('confirmcurve')
ax[1].imshow(dog_hog_img, cmap='gray')
ax[1].set_title('hog')
plt.show()
```

(1) As well as the transformer component code, which is necessary in order to complete the compression of the inputted images into numbers that can be understood by the SGD classifier:

```
class RGB2GrayTransformer(BaseEstimator, TransformerMixin):
    """
    Convert an array of RGB images to grayscale
    """

    def __init__(self):
        pass

    def fit(self, X, y=None):
        """returns itself"""
        return self

    def transform(self, X, y=None):
        """perform the transformation and return an array"""
        return np.array([skimage.color.rgb2gray(img) for img in X])


class HogTransformer(BaseEstimator, TransformerMixin):
    """
    Expects an array of 2d arrays (1 channel images)
    Calculates hog features for each img
    """

    def __init__(self, y=None, orientations=9,
                 pixels_per_cell=(8, 8),
                 cells_per_block=(3, 3), block_norm='L2-Hys'):
        self.y = y
        self.orientations = orientations
        self.pixels_per_cell = pixels_per_cell
        self.cells_per_block = cells_per_block
        self.block_norm = block_norm

    def fit(self, X, y=None):
        return self

    def transform(self, X, y=None):

        def local_hog(X):
            return hog(X,
                       orientations=self.orientations,
                       pixels_per_cell=self.pixels_per_cell,
                       cells_per_block=self.cells_per_block,
                       block_norm=self.block_norm)

        try: # parallel
            return np.array([local_hog(img) for img in X])
        except:
            return np.array([local_hog(img) for img in X])
```

d) Finally, I added in the SGD classifier code itself, which trains the classifier to check the now compressed images for a difference in data points (in the case of the light curve, this would ideally be the dip of the graph):

```
#Phase III.5: SGD Classifier:

sgd_clf = SGDClassifier(random_state=42, max_iter=1000, tol=1e-3)
sgd_clf.fit(X_train_prepared, y_train)

SGDClassifier(random_state=42)
```

2. Phase **four** involves the testing of the algorithm itself, which I was able to implement relatively easily as seen below, as it is only verifying and comparing what has already been done to the data:

```
#Phase IV: Testing & Analysis:

X_test_gray = grayify.transform(X_test)
X_test_hog = hogify.transform(X_test_gray)
X_test_prepared = scalify.transform(X_test_hog)

y_pred = sgd_clf.predict(X_test_prepared)
print(np.array(y_pred == y_test)[:25])
print('')
print('Percentage correct: ', 100*np.sum(y_pred == y_test)/len(y_test))
```

C. <u>Next Steps:</u>

1. Begin attempts at running the code; check for errors and remedy issues as they arise in the algorithm.
2. Add more in-depth analysis code to attain a wider scope of data from each test trial.
3. Continue research into GNNs and browse for examples, especially with graphs similar to those in my datasets
    a) Consult the literature found previously - which method did they use, if that information is readily available?
    b) Decide, after the further research mentioned, whether or not to create a different sort algorithm using a GNN and PyTorch.

X. 2.28-3.5.25 Entry: Further Research on Graph Neural Networks and Modification of Light Curve Sort Algorithm:

A. Remedying Errors in Sort Algorithm A (HOG-SVM & SGD - typical image classifier):

1. Syntax error with regard to file acquisition (the path I entered to import the light curve graph files):

```
SyntaxError: (unicode error) 'unicodeescape' codec can't decode bytes in position 2-3: truncated \UXXXXXXXX
escape
```

   a) Solved by utilizing escape characters (*r* and additional backslashes - this error actually comes up often in Java, so it was not as difficult to resolve).

2. Attribute error with regard to numpy (as used in the pre-processing of data):

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[1], line 27
     25         img = imread(img_path)
     26         resize(img, (15, 15))
---> 27         data.append(img.flatted())    #required because I used a certain basic classifier
     28         labels.append(category_idx)
     30 data = np.asarray(data)

AttributeError: 'numpy.ndarray' object has no attribute 'flatted'
```

   a) It was the case that I had written “flatted” when the method was called “flatten.” Once I realized this, I was able to resolve the issue.

3. Value error concerning setting an array element with a sequence of numbers rather than one specific number (also in the pre-processing step):
   a) I first attempted to solve this issue by defining the type of data to be contained within the arrays I was defining:

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[3], line 31
     28         labels.append(category_idx)
     30 Data_type = int
---> 31 data = np.asarray(data, dtype = Data_type)
     32 labels = np.asarray(labels, dtype = Data_type)
     34 #Phase II: Splitting data for test/training sets:

ValueError: setting an array element with a sequence. The requested array has an inhomogeneous shape after 1 dimensions. The detected shape was (16,) + i
nhomogeneous part.
```

b) However, this did not work, and so I attempted to solve the error via an alternative method: I tried to match the data types of both rather than simply setting one equal to one specific data type arbitrarily (as I had done above).

```
Data_type = tuple
data = np.asarray(data, dtype = Data_type)
labels = np.asarray(labels, dtype = Data_type)
```

c) After several trial-and-error attempts, I recalled that the data type of the information I was combining was *tuple* (an ordered, immutable list of numbers). As a result, I set the data type to tuple, and the error was resolved.

4. I resolved a few other errors, though there are still 1-3 that might arise as I continue to manipulate the code.

B. <u>Additional Research on GNNs and Building Sort Algorithm B (GNN & PyTorch):</u>

1. Research and Comprehension of GNNs:
   a) Helpful and Informative video on GNNs (includes an explanation of each component and how those parts contribute to a high-accuracy algorithm;
      (1) See link: https://www.youtube.com/watch?v=ZmYsiBtpDjk
      (2) See related code file, containing all of the information necessary to fully implement the GNN presented in the above video: https://xbe.at/index.php?filename=Improving+Graph+Classification+Accuracy+with+Python.md

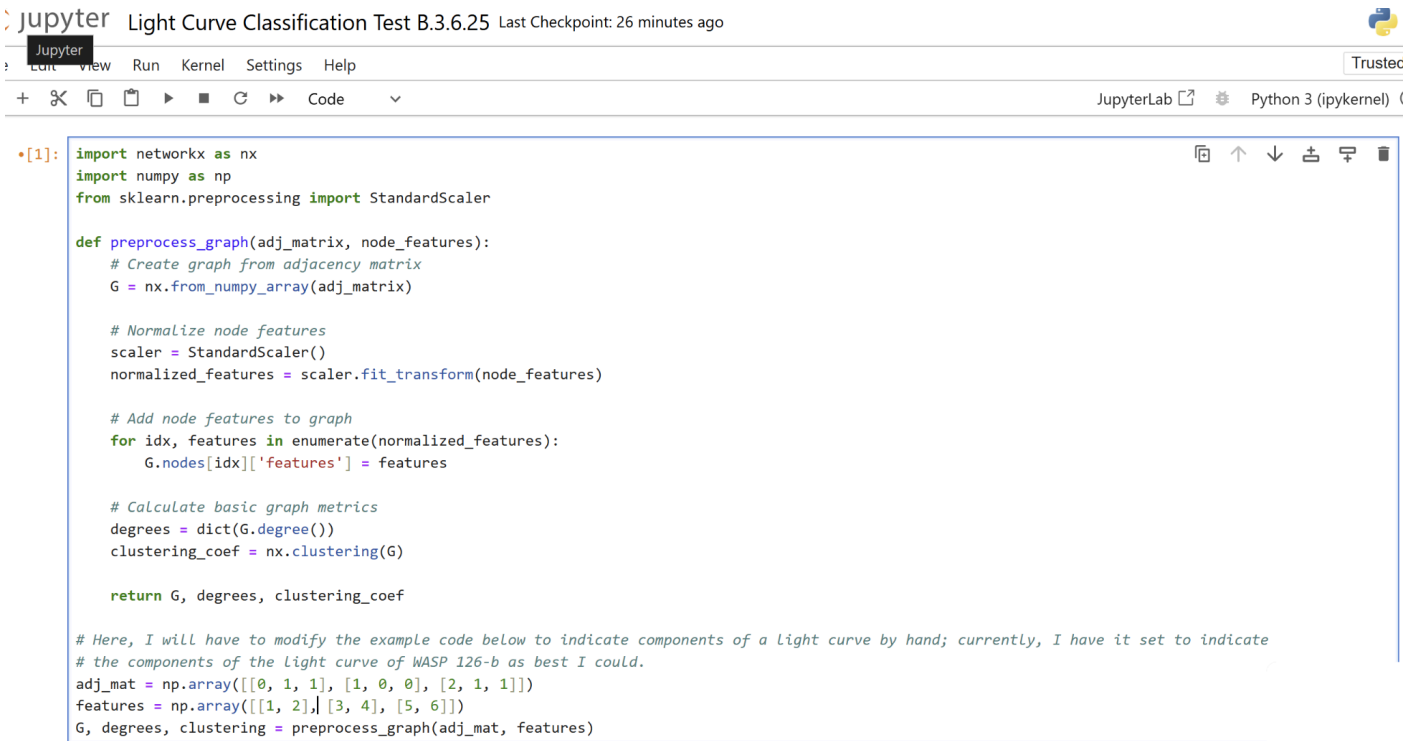
      (3) NOTE: this video and the accompanying code offer a tangible example of how one can combine the components of Scikit-Learn with the GNN classifying of PyTorch into one complete algorithm.
         (a) This appears to be successful based on the information presented and resolves my issue of

having to completely convert my knowledge from one machine learning library to another.

(b) As a result, I will likely pursue a similar path for my Sort Algorithm B, based on a GNN classifier.

2. Constructing Phases 1 (Pre-processing of data) and II (Training & Test Split / Preparation of datasets) of the new GNN-based algorithm, Sort Algorithm B:

   a) Here is the pre-processing (Phase I) code I utilized for this new algorithm:



(1) NOTE: It is only designed for one light curve as of this moment, for I need to find a better way of producing an adjacency matrix using code. Currently, despite my research into it, I have been unable to find any way of determining an adjacency matrix other than doing it by hand.

   (a) SUGGESTION: Perhaps I could use Matplotlib to mimic what I do by hand to make this matrix by selecting points on the graph (using similar code as in my earlier light curve processing algorithm?)

(b) SUGGESTION TAKEN: I will try removing the need for an adjacency matrix entirely by replacing its position in the initial definition of graph G with a file call to a light curve, as shown in the modified code below:

```python
import networkx as nx
import numpy as np
from sklearn.preprocessing import StandardScaler

def preprocess_graph(file, node_features):
    G = file

    # Normalize node features
    scaler = StandardScaler()
    normalized_features = scaler.fit_transform(node_features)

    # Add node features to graph
    for idx, features in enumerate(normalized_features):
        G.nodes[idx]['features'] = features

    # Calculate basic graph metrics
    degrees = dict(G.degree())
    clustering_coef = nx.clustering(G)

    return G, degrees, clustering_coef

# Here, I have switched out the adjacency matrix:
features = np.array([[1, 2], [3, 4], [5, 6]])
G, degrees, clustering = preprocess_graph(r'C:\Users\bwlam\OneDrive\Documents\SRP 2025 - Light Curve Datasets\Confirmed\wasp126b.png', features)

# However, this still presents the issue of the features matrix. I will need to research that further.
```

(2) The following is code that can be utilized to extract graph features and produce the feature matrix, but it only functions provided that there is a graph, G, already in existence with an adjacency matrix:

```python
# Features Algorithm:
def extract_curve_features(G):
    features = {}

    # Structural features
    features['num_nodes'] = G.number_of_nodes()
    features['num_edges'] = G.number_of_edges()
    features['density'] = nx.density(G)

    # Spectral features
    laplacian = nx.normalized_laplacian_matrix(G)
    eigvals = np.linalg.eigvals(laplacian.toarray())
    features['spectral_radius'] = max(abs(eigvals))

    # Centrality measures
    features['avg_betweenness'] = np.mean(list(nx.betweenness_centrality(G).values()))
    features['avg_closeness'] = np.mean(list(nx.closeness_centrality(G).values()))

    return features

curve_features = extract_curve_features(G)
```

b) In terms of training / test dataset splits, the code is essentially the same as it was in Sort Algorithm A.

C. Next Steps:

1. Now that it has been debugged, I will more fully test Sort Algorithm A (SGD) for its functionality and (hopefully) accuracy.
2. In addition, I will continue working on Sort Algorithm B (GNN) based on the research I conducted for this cycle, namely by finding a new way to produce feature matrices and by completing Phase 3 (and possibly 4).
    a) That way, I can begin editing, adapting, and testing Sort Algorithm B sooner than later and explore the likely more accurate GNN pathway going forward.
3. I will also be prepared should any further response be made to my contact by professors and/or their grad students.

XI. 3.14-3.20.25 Entry: Completion of Graph Neural Network, Continued Research into Fixing Bugs, & Preparation for Testing:

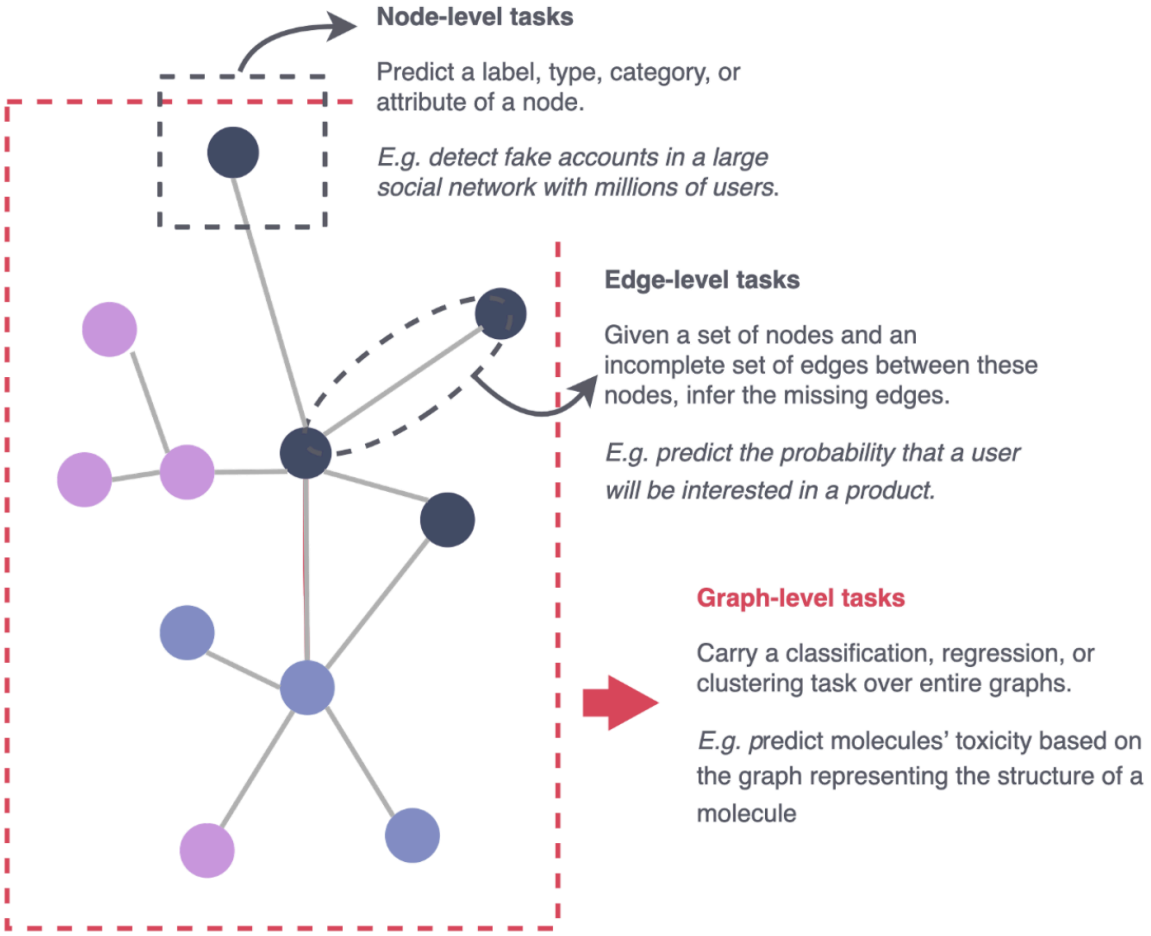
A. Completing Sort Algorithm B (Graph Neural Network):

1. This past cycle, I put what Dr. Matone and I discussed previously into practice by focusing my efforts on the algorithm with the most promising chance of success: the GNN, referred to above as Sort Algorithm B.

2. In order to do so, I conducted a significant amount of additional research into GNNs, referencing a variety of sites and online documentation in an attempt to find what would work best for me going forward. In my research, I learned the following (which I found to be very useful):
    a) I should treat each light curve graph as a **scatter plot**, for that is the easiest way to convert them into GNN graphs, as discussed below. Also, light curves, when broken down fundamentally, are simply scatter plots where the points dip for a given period of time.

    b) GNNs turn input information into graphs with **nodes** and **edges**:
        (1) Nodes are the individual points within the scatterplot
        (2) Edges are the connections between the nodes
            (a) In this case, because I am looking at scatter plots (light curves), there is no direct connection (line) between points; however, edges can still record the distance from one node to nearby nodes, a helpful

tool for when looking at light curves, especially for classification.



GNN Infographic - Source: https://blog.dataiku.com/graph-neural-networks-part-three
Useful YouTube Educational Playlist on GNNs:
https://www.youtube.com/playlist?list=PLV8yxwGOxvvoNkzPfCx2i8an--Tkt7O8Z

c) Each node can have a set of **features** that saves information such as the x and y coordinates of a point on a given light curve scatter plot. Those features, as noted in the code development for the previous cycle, are stored in a matrix that is then compared to the matrices of other nodes to check for similarities.

3. With this research in mind, I edited my existing code and added to it to complete my GNN (see phase-by-phase breakdown below) with the following functions:
    a) Convert a light curve into a graph of nodes and edges
    b) Coding and training a scatter plot classifier by comparing node features and edge data from one curve to another
    c) Outputting results in a comprehensible format designed to help me improve the code in the future.

4. Completing **Phase 3** (GNN Classifier):
    a) As I learned through my research, the heart of a GNN algorithm, the classifier, functions by first learning the typical node and edge data of a confirmed and an unconfirmed exoplanet light curve via the training dataset.
    b) Then, when given a light curve at random, by amassing node features and comparing them to its saved knowledge of what the features of a confirmed and unconfirmed light curve look like, the algorithm determines whether or not the light curve is likely of a confirmed exoplanet or of some other astrophysical phenomenon.

This first half of the classification process is coded to the left (note the use of PyTorch; now, my algorithm is a hybrid of Scikit-Learn and PyTorch methods):

```
#Phase 3: Classifier
class GraphClassifier(nn.Module):
    def __init__(self, input_dim, hidden_dim, num_classes):
        super(GraphClassifier, self).__init__()

        self.conv1 = gnn.GCNConv(input_dim, hidden_dim)
        self.conv2 = gnn.GCNConv(hidden_dim, hidden_dim)
        self.pool = gnn.global_mean_pool
        self.fc = nn.Linear(hidden_dim, num_classes)

    def forward(self, x, edge_index, batch):
        # Message passing layers
        x = torch.relu(self.conv1(x, edge_index))
        x = torch.relu(self.conv2(x, edge_index))

        # Graph-level pooling
        x = self.pool(x, batch)

        # Classification
        return self.fc(x)

# Model initialization
model = GraphClassifier(input_dim=10, hidden_dim=64, num_classes=2)
```

c) *Basis of code - credit to* [*https://xbe.at/index.php?filename=Improving+Graph+Classification+Accuracy+with+Python.md*](https://xbe.at/index.php?filename=Improving+Graph+Classification+Accuracy+with+Python.md)

d) In order for the above code to work as accurately as possible, the average of confirmed and unconfirmed node features must be produced via graph *pooling*, as coded below:

```
#Phase 3, Pt. 2: Pooling:
class HierarchicalPooling(nn.Module):
    def __init__(self, in_channels, ratio=0.5, min_score=None):
        super(HierarchicalPooling, self).__init__()

        self.topk_pool = gnn.TopKPooling(
            in_channels,
            ratio=ratio,
            min_score=min_score
        )
        self.sag_pool = gnn.SAGPooling(
            in_channels,
            ratio=ratio,
            min_score=min_score
        )

    def forward(self, x, edge_index, batch):
        # TopK pooling
        x1, edge_index1, _, batch1, _, _ = self.topk_pool(
            x, edge_index, None, batch
        )

        # SAG pooling
        x2, edge_index2, _, batch2, _, _ = self.sag_pool(
            x, edge_index, None, batch
        )

        # Combine pooling results
        x_combined = torch.cat([x1, x2], dim=-1)
        return x_combined, batch1
```

5. Completing **Phase 4** (Printing test data, predictions, and accuracy numbers to the user):
    a) In order to report the test data, I used the following code, supplemented by a plot generator that transforms the outputted data into a legible graph of predicted vs. actual:

    b) Useful video & attached code - credit to [*https://www.youtube.com/watch?v=0YLZXjMHA-8&list=PLV8yxwGOxvvoNkzPfCx2i8an--Tkt7O8Z&index=3*](https://www.youtube.com/watch?v=0YLZXjMHA-8&list=PLV8yxwGOxvvoNkzPfCx2i8an--Tkt7O8Z&index=3)

```
#Phase 4: Testing & Reporting:
test_batch = next(iter(test_loader))
with torch.no_grad():
    test_batch.to(device)
    pred, embed = model(test_batch.x.float(), test_batch.edge_index, test_batch.batch)
    df = pd.DataFrame()
    df["y_real"] = test_batch.y.tolist()
    df["y_pred"] = pred.tolist()
df["y_real"] = df["y_real"].apply(lambda row: row[0])
df["y_pred"] = df["y_pred"].apply(lambda row: row[0])
df

#Phase 4, Add: Plot Creation:
plt = sns.scatterplot(data=df, x="y_real", y="y_pred")
plt.set(xlim=(-7, 2))
plt.set(ylim=(-7, 2))
plt
```

B. Other Preparations Before Testing:

1. *Expanding Confirmed Light Curve dataset:*
    a) In preparation for testing, now that my algorithm has been finished, I set out to use my original code to extract additional confirmed exoplanet light curves from the MAST database.

    b) I did this so that my algorithm would be easier to train and test. Further, the larger dataset improves the validity of my program, methods, and conclusion from a scientific perspective by showing that what I am trying to prove, if possible, is not simply the result of a selective, biased, or limited sample size.

2. *Debugging GNN Code:*
    a) Similar to the work I did for last cycle, after completing the GNN to a relatively acceptable degree, I began running the code and solving issues as they came up. This included some of the following:

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[1], line 24
     22 # Here, I have switched out the adjacency matrix:
     23 features = np.array([[1, 2], [3, 4], [5, 6]])
---> 24 G, degrees, clustering = preprocess_graph(r'C:\Users\bwlam\OneDrive\Documents\SRP 2025 - Light Curve Datasets\Confirmed\wasp126b.png', features)
     26 # However, this still presents the issue of the features matrix. I will need to research that further.
     27
     28 # Features Algorithm:
     29 def extract_curve_features(G):

Cell In[1], line 14, in preprocess_graph(file, node_features)
     12 # Add node features to graph
     13 for idx, features in enumerate(normalized_features):
---> 14     G.nodes[idx]['features'] = features
     16 # Calculate basic graph metrics
     17 degrees = dict(G.degree())

AttributeError: 'str' object has no attribute 'nodes'
```

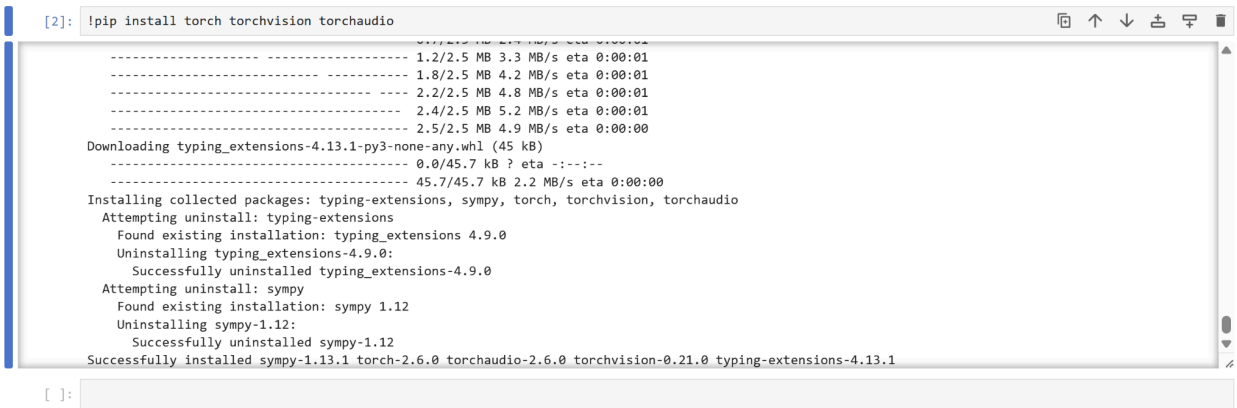
C. Next Steps:

1. Now that I have focused my efforts solely on Light Curve Sort Algorithm B (GNN) and completed it, I plan to dedicate time each day after school to **training, testing, and refining** the algorithm. In doing so, I will continue to adapt portions of the code to my specific needs.
    a) This way, not only can I begin to get firm and clear results, but I can also get a better understanding of how GNNs work in practice so that I can improve my own.
    b) I also can add supplementary code to the end of the algorithm so that it can better return data on its functioning and efficiency, data that can easily be understood graphically.

XII. 3.28-4.14.25 Entry: Testing GNN Algorithm & Beginning Work on Symposium Files for Presentation:

A. GNN Testing and Optimization:

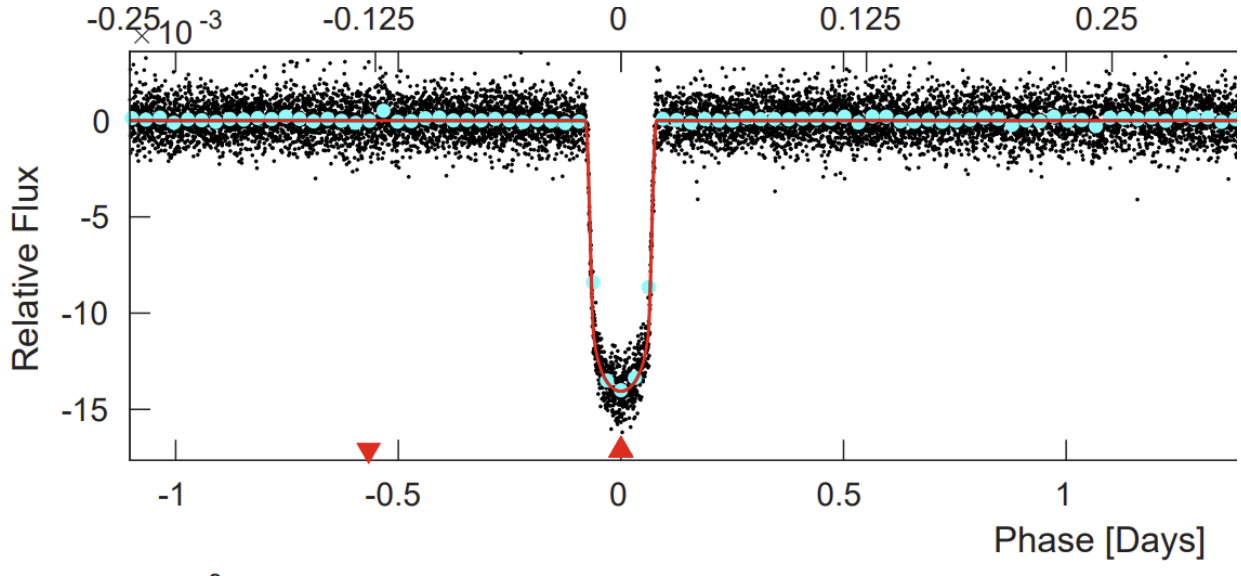
1. In my initial tests, I encountered a few errors with the code that I had to remedy. These ranged from a mislocation of one of the machine learning libraries (PyTorch) to smaller syntax issues.
    a) In doing so, I had to utilize the command prompt in checking my installation of and reinstalling both PyTorch and its sub-library torch-geometric.
    b) This included installing them directly into Jupyter, not just my own computer, using the *!pip install torch torchvision torchaudio* command and a similar one for PyTorch geometric (torch_geometric).

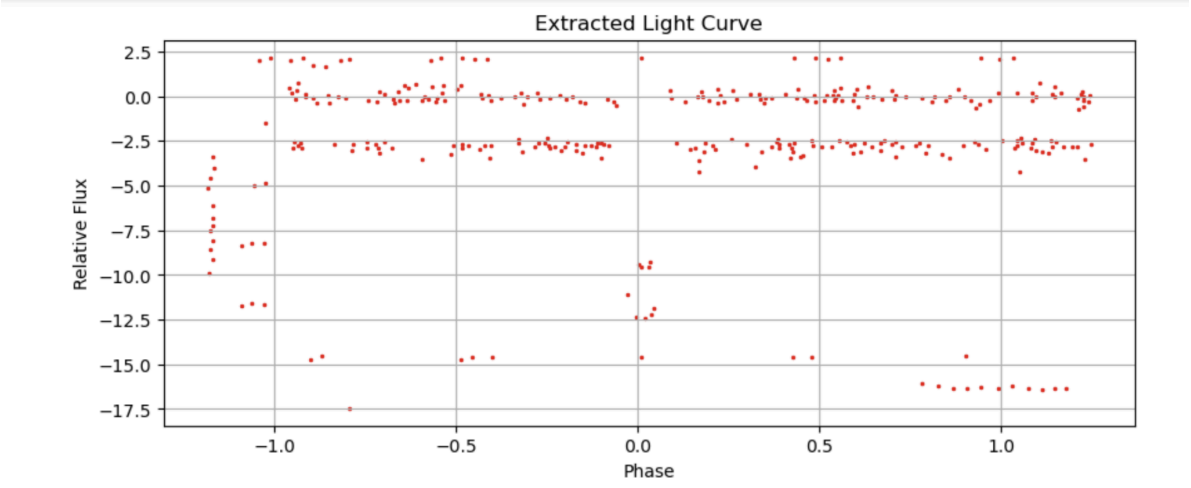


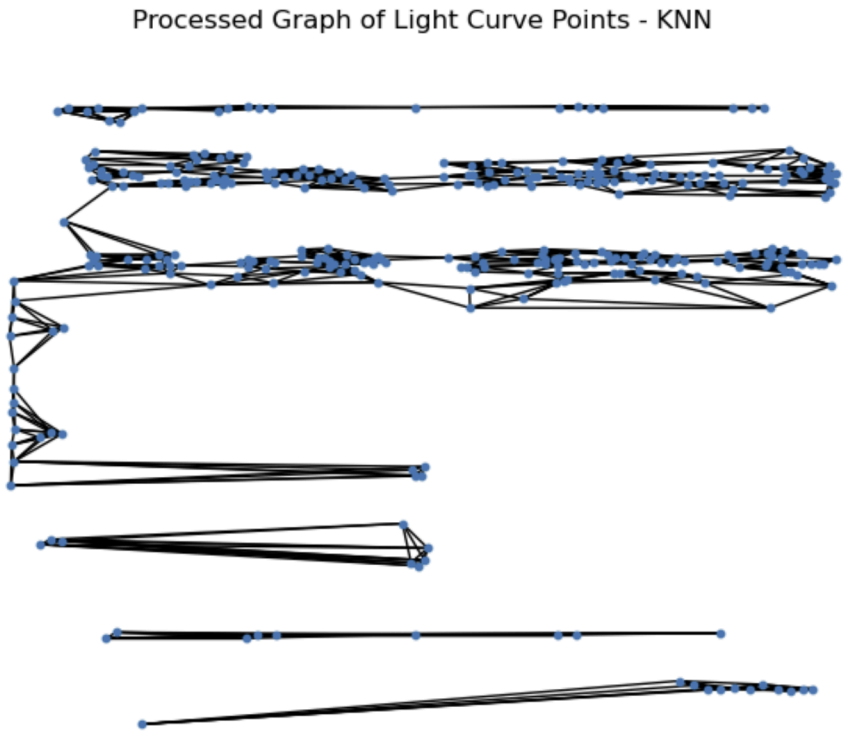
2. In addition, I also had to edit the portion of code in which I imported a light curve from my hard drive (from the datasets I have created and shown above) to ensure that it was both in the right location within the program and that it was functioning properly.

B. Complete Debugging (conducted 4/13-4/14/25):

1. In my process of debugging, I compiled a separate pre-processing algorithm that, in contrast from the previous ones I attempted, functioned fairly well. By doing so, I believe I also produced two figures (one being a plot of extracted data from a light curve image and the other being the edges and nodes created from that data, ready to be inputted into the heart of my GNN algorithm- see both below).







C. Completing the Initial Stages of Presentation Documents:

1. Project Title: *Leveraging Deep Learning to Classify TESS Light Curves as a Novel Method for Confirming New Exoplanets*

2. Brief Description (Now normalized, **pending Dr. Matone's approval**):
    a) Using Python, I designed an algorithm to sort light curves observed by TESS (Transiting Exoplanet Survey Satellite). I did so to test the accuracy of such an algorithm in determining whether or not a given light curve was produced by an exoplanet or by some other celestial object and experimented with the use of this algorithm to identify new exoplanets from undetermined TESS data.

3. Drafting Introduction and Abstract:
    a) Introduction:
        (1) Since the launch of the TESS (Transiting Exoplanet Survey Satellite) mission in 2018, the probe has observed thousands of different celestial objects using the transiting detection method. When an object passes between its host star and the Earth, it transits across the star, causing its

luminosity to dip slightly and allowing TESS to produce a graph known as the light curve. Due to the fact that the light curve often provides more information about the host star than it does the orbiting planets themselves, it does not immediately show whether or not the detected object is an exoplanet or some other astronomical body (Yang et al. 2024). As a result, determining a light curve to be from an exoplanet can involve additional testing and time. To streamline that process to suit the large quantities of data amassed by TESS and similar observatories, researchers have begun using deep learning, a kind of machine learning (AI) model, to sort light curves (Tey et al. 2023).

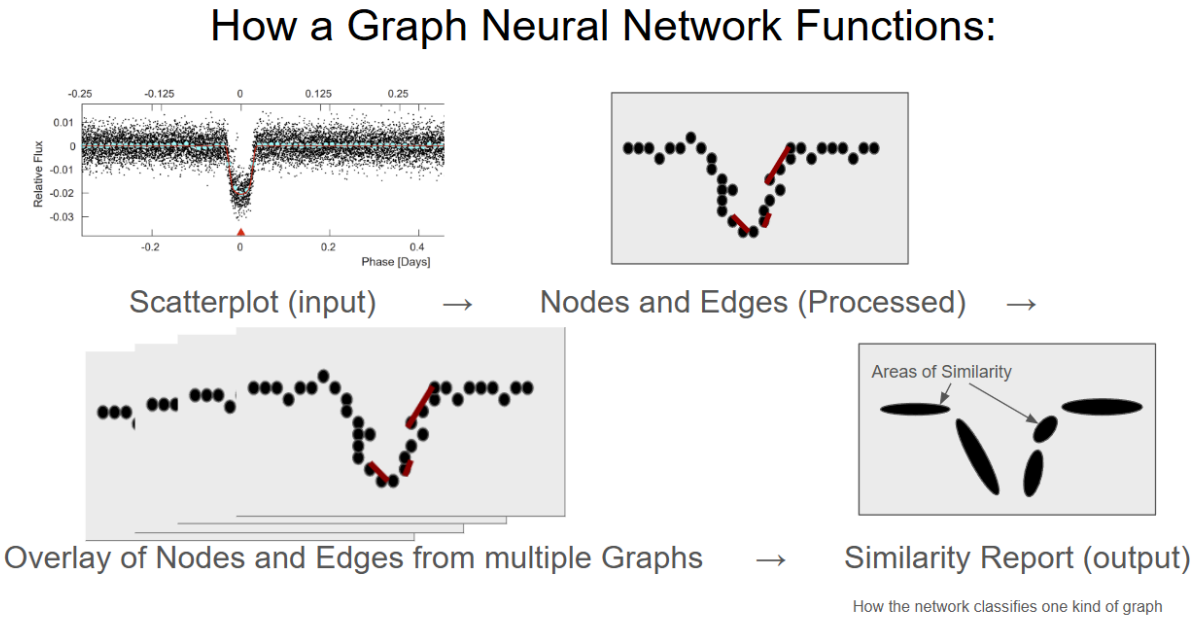
(2) However, this classification of light curves to identify exoplanets has often only been done reliably with convolutional neural networks (CNNs) that rely on whole image sorting (Cuéllar et al. 2022). Therefore, I attempted to test a different approach using a graph neural network (GNN) to break light curves down into components and classify them either as exoplanet transits or so-called 'false positives' based on the similarities between those components.

b) Abstract:

(1) The TESS observatory was launched to discover new exoplanets in nearby star systems via the transiting method, in which a planet is detected when it passes in front of its host star, temporarily decreasing the star's brightness and causing a dip in the graph of brightness vs. time known as a light curve. Yet, non-planetary objects can create light curves as well, and thus identifying data to actually be from an exoplanet requires additional time and resources. To optimize the exoplanet confirmation procedure and thus process the large amount of TESS data faster and more reliably, I designed a graph neural network (GNN) to classify light curves using python in Jupyter Notebook. In the past, convolutional neural networks (CNNs) have been used often for light curve classification, so I endeavored to study the largely uninvestigated potential of GNNs. By first writing a program to extract TESS light curves from the

MAST (Mikulski Archive for Space Telescopes) database, I was able to compile a dataset of light curves of confirmed exoplanets. Combining those graphs with unconfirmed light curves taken from the ExoFOP (Exoplanet Follow-Up Observing Program) database, I assembled a total collection of roughly 50 light curves. After the graphs were pre-processed and the algorithm was trained, I found the GNN to have a reliable accuracy for determining whether or not a given light curve was from a confirmed exoplanet or another stellar object. Ultimately, by establishing the relatively high efficiency and accuracy of a GNN when classifying light curves, I have proved an alternative way to further optimize humanity's discovery of exoplanets from TESS data.

4. Plan for graphics to include:
    a) Perhaps I should add the GNN infographic shown above as one of my images so that viewers have a visual aid to understand my algorithm when I am introducing it to them during the Symposium.
        (1) I could also, instead, use the flowchart below that I created as a more simplified version of the infographic in entry XI to visually show how a GNN functions:



b) I believe I also will include a sample light curve from each dataset: One from the confirmed exoplanets or planetary candidates set and another from the non-confirmed exoplanets or 'false positives' dataset. That way, viewers will be able to get an understanding of what the samples I am using look like, as well as an idea of what a light curve is.

D. Next Steps (Over Break):

1. The Symposium is the Thursday we come back from Easter recess, so over the next few days and the break that follows, I plan to:
    a) Finish debugging the GNN algorithm
    b) (Attempt to) train and test the algorithm, producing plottable results
    c) Complete my abstract, methods, results (once they come in), and discussion slides for my Symposium poster board
    d) Begin gluing together and preparing said poster board

XIII. 4.17-4.30.25 Entry: Compiling the Final Iteration of the GNN Algorithm and Completing SRP Symposium Documents:

A. To-Do Before Symposium:

- ☑ ~~Verify Unconfirmed Dataset (see section B below)~~
- ☑ ~~Complete Datasets~~
    - ☑ ~~Confirmed~~
    - ☑ ~~Unconfirmed~~
- ☑ ~~Complete written poster-board documents~~
    - ☑ ~~Abstract~~
    - ☑ ~~Introduction~~
    - ☑ ~~Materials~~
    - ☑ ~~Methods~~
    - ☑ ~~Results~~
    - ☑ ~~Discussion~~
    - ☑ ~~Conclusion & Future Steps~~
    - ☑ ~~Acknowledgements~~
- ☑ ~~Complete the GNN~~
- ☑ ~~Train the GNN~~
- ☑ ~~Test the GNN~~

B. Dataset Updates:

1. IMPORTANT:
2. As I was going through and adding more light curves to my datasets, I had the thought to supplement the unconfirmed set with asteroid light curves that I found in the DAMIT database (linked here) and supernova light curves found in the CfA Supernova Data Archive (linked here).
    a) However, I think that for the Science Symposium, I will progress forward with my current sources and datasets (MAST and ExoFOP) with the distinction between objects known to be planets and objects (TOIs) that are not yet confirmed as planets.
    b) I will need to go through my dataset to ensure that the light curves labeled as unconfirmed are, in fact, unconfirmed. I will use a list on the NASA Exoplanet Archive (linked here) to cross-reference my dataset to ensure that all light curves in the unconfirmed set are, in fact, unconfirmed.
        (1) TOI-101.01 is confirmed
        (2) TOI-102.01 is confirmed
        (3) Ultimately, excluding one ambiguous case, all of the light curves that were originally in my 'unconfirmed' dataset were actually confirmed. Thus, I made the necessary changes and sought out a new source of unconfirmed light curves: false positives.

3. Verified Unconfirmed (as classified as 'false positives,' which means that they are **not** exoplanets, by this academic journal: Hord et. al. 2023, see figure 7 on page 27 of the pdf):
    a) TOI-1022.01 (light curve not on ExoFOP)
    b) TOI-864.01 (light curve successfully found)
    c) TOI-212.01 (light curve successfully found)
    d) TOI-706.01 (light curve successfully found)
    e) TOI-539.01 (light curve successfully found)
    f) TOI-906.01 (light curve successfully found)
    g) TOI-1355.01 (light curve successfully found)

4. I then found the light curves for these unconfirmed TOIs on the ExoFOP database. Note that the database itself also had notes on the above unconfirmed TOIs describing their unconfirmed status and corroborating the results of the Hord et. al. paper.

5. The only potential issue is that now that I have found verifiably unconfirmed light curves, I still have the challenge of processing them. They do not appear to be similar to the confirmed curves, as they have additional data on them, potentially making my classifications invalid.
    a) I believe I can solve this problem by writing in an "if" statement such that the algorithm processes png light curve images in the case of the confirmed curves and processes .csv files (processed graph data arrays, which hopefully will not be different than the confirmed .csvs) directly downloaded from ExoFOP in the case of unconfirmed curves.
    b) I will likely include a bit about this in the discussion section of my presentation.

C. Symposium writing can be found on the attached slideshow, which consists of all sections to be printed, cut out, and glued to poster board by Thursday afternoon.
    1. Link: https://docs.google.com/presentation/d/1cq8qByvZk1MtWzfbnxIwTByLX2-2uPuXabxjm-FLi2M/edit?usp=drivesdk

New Features:

- Added a "dropout" to prevent the GNN from relying on one feature too much for classification, to improve accuracy

---

XIV. 5.8-5.16.25 Entry: Expanding the Datasets & Optimizing GNN Classifier for Small Dataset Sizes to Improve Accuracy:

A. Dataset Expansions; Light Curves Added:
    1. Unconfirmed:
        a) TOI-1967.01
        b) TOI-1770.01
        c) TOI-1254.01
        d) And others.

    2. Testing Dataset:
        a) TOI-1254.01 (supposed to be unconfirmed)

B. GNN Optimizations (smaller-scale edits to the algorithm to improve accuracy and fix classification mistakes):
    1. Adding a Cost Function (aka loss function) → **Cross-Entropy Loss**:

a) REFERENCE: article explaining different kinds of cost functions, including the one I implement here (a cross-entropy loss, designed specifically for binary classifications such as mine)
b) Code (added to algorithm after graph list is produced):

```
# --- Calculate class weights for weighted loss ---
labels = [data.y.item() for data in graphs]
class_counts = np.bincount(labels)
total = sum(class_counts)
class_weights = [total / c for c in class_counts]
class_weights_tensor = torch.tensor(class_weights, dtype=torch.float).to(device)
```

(1) This code checks the labels from the pre-processed graphs in the *graphs* list and assigns classification weight to them. That way, if the dataset consists of more confirmed than unconfirmed light curves, instead of the classifier being more inclined to guess that a new light curve is confirmed (simply because there are more of them in the training set), it will attribute more weight and consideration to the unconfirmed classification. This will hopefully improve classification accuracy.

(2) Initially, I encountered an error, but, after moving the definition of the 'device' variable, it was resolved.

2. Understanding and Producing a **Confusion (Error) Matrix**:
   a) REFERENCE: article explaining the parts and workings of a confusion matrix (essentially the different data it shows comparing actual classifications with predicted ones)

XV. 5.17-6.2.25 Entry: Fixing Training Split, Making Final Slideshow, & Additional Tests of the Proper Algorithm / GNN Pipeline:

A. Fixing Improper Class Split in the Training Code:

1. After running several tests of the algorithm, I found that it predicted each light curve I fed it to be from a confirmed exoplanet (even when I fed it curves from unconfirmed objects). To check the issue, I added a section of code that printed the distribution of unconfirmed and confirmed light curves in the training dataset, and I found that the code was dividing by 0 in one location due to the fact that no unconfirmed light curves were making it into the training dataset.
    a) This was due to an error in the labeling of graphs after preprocessing using the **is_confirmed_exoplanet(filepath)** function, which is designed to label graphs as either confirmed or unconfirmed from their filepath in my computer's hard drive.

```
def is_confirmed_exoplanet(filepath):
    folder = os.path.basename(os.path.dirname(filepath)).lower()
    return folder == "confirmed"
```

    b) I fixed the issue by adding a new line of code to the is_confirmed_exoplanet function (see completed version above), ensuring that it processed the file path of each light curve correctly.

2. When I performed the first test with this error resolved, the code worked properly for the first time, training itself over 50 epochs and improving its accuracy over those iterations (unlike the previous trials where it immediately achieved 100% accuracy at the first epoch). In addition, it finally classified an unconfirmed light curve correctly in the first test I conducted following this fix (see subsection B below).

See next page

B. First Pipeline Tests (Now that the error is fixed and results are valid):

1. No. 1:

a) Results:

```
Epoch 1, Train Loss: 2.7570, Val Acc: 0.33
Epoch 2, Train Loss: 2.6226, Val Acc: 0.33
Epoch 3, Train Loss: 2.7788, Val Acc: 0.33
Epoch 4, Train Loss: 2.5459, Val Acc: 0.33
Epoch 5, Train Loss: 2.6679, Val Acc: 0.33
Epoch 6, Train Loss: 2.5023, Val Acc: 0.33
Epoch 7, Train Loss: 2.4700, Val Acc: 0.33
Epoch 8, Train Loss: 2.4803, Val Acc: 0.33
Epoch 9, Train Loss: 2.4628, Val Acc: 0.33
Epoch 10, Train Loss: 2.4156, Val Acc: 0.33
Epoch 11, Train Loss: 2.4868, Val Acc: 0.33
Epoch 12, Train Loss: 2.4845, Val Acc: 0.33
Epoch 13, Train Loss: 2.4704, Val Acc: 0.33
Epoch 14, Train Loss: 2.3800, Val Acc: 0.50
Epoch 15, Train Loss: 2.4146, Val Acc: 0.83
Epoch 16, Train Loss: 2.3395, Val Acc: 0.83
Epoch 17, Train Loss: 2.3232, Val Acc: 0.83
Epoch 18, Train Loss: 2.2761, Val Acc: 0.83
Epoch 19, Train Loss: 2.2084, Val Acc: 0.50
Epoch 20, Train Loss: 2.1192, Val Acc: 0.33
Epoch 21, Train Loss: 2.3098, Val Acc: 0.33
Epoch 22, Train Loss: 2.0634, Val Acc: 0.33
Epoch 23, Train Loss: 2.0413, Val Acc: 0.83
Epoch 24, Train Loss: 1.9785, Val Acc: 0.83
Epoch 25, Train Loss: 2.0263, Val Acc: 0.83
Epoch 26, Train Loss: 1.9742, Val Acc: 0.83
Epoch 27, Train Loss: 1.8675, Val Acc: 1.00
Epoch 28, Train Loss: 1.8236, Val Acc: 1.00
Epoch 29, Train Loss: 1.7543, Val Acc: 1.00
Epoch 30, Train Loss: 1.7064, Val Acc: 1.00
Epoch 31, Train Loss: 1.6549, Val Acc: 1.00
Epoch 32, Train Loss: 1.5614, Val Acc: 1.00
Epoch 33, Train Loss: 1.5215, Val Acc: 1.00
Epoch 34, Train Loss: 1.4370, Val Acc: 1.00
Epoch 35, Train Loss: 1.4133, Val Acc: 1.00
Epoch 36, Train Loss: 1.3987, Val Acc: 1.00
Epoch 37, Train Loss: 1.2958, Val Acc: 1.00
Epoch 38, Train Loss: 1.2090, Val Acc: 1.00
Epoch 39, Train Loss: 1.1889, Val Acc: 1.00
Epoch 40, Train Loss: 1.1445, Val Acc: 1.00
Epoch 41, Train Loss: 1.0833, Val Acc: 1.00
Epoch 42, Train Loss: 0.9700, Val Acc: 1.00
Epoch 43, Train Loss: 1.0182, Val Acc: 1.00
Epoch 44, Train Loss: 0.9939, Val Acc: 1.00
Epoch 45, Train Loss: 0.8641, Val Acc: 1.00
Epoch 46, Train Loss: 0.8567, Val Acc: 1.00
Epoch 47, Train Loss: 0.8706, Val Acc: 1.00
Epoch 48, Train Loss: 0.8183, Val Acc: 1.00
Epoch 49, Train Loss: 0.8534, Val Acc: 1.00
Epoch 50, Train Loss: 0.7501, Val Acc: 1.00
Best validation accuracy: 1.00
Train class distribution: Counter({1: 9, 0: 6})
Val class distribution: Counter({1: 4, 0: 2})
              precision    recall  f1-score   support

 Unconfirmed       1.00      1.00      1.00         2
   Confirmed       1.00      1.00      1.00         4

    accuracy                           1.00         6
   macro avg       1.00      1.00      1.00         6
weighted avg       1.00      1.00      1.00         6

Validation Accuracy: 1.0
Prediction for TOI-1254.01.png: Unconfirmed
```

These results show that the algorithm is now working the way it was intended: processing each curve at a time while training and improving accuracy in the process (not just getting a 1.00 on accuracy right away).

Test I (TOI-1254.01): Unconfirmed input → Unconfirmed output
Test II (TOI-1873.01): Unconfirmed input → Unconfirmed output
Test III (TOI-1853.01): Confirmed input → Unconfirmed output *INCORRECT*
Test IV (TOI-871.01): Confirmed input → Unconfirmed output *INCORRECT*

As seen by the four tests above, it is now working in the opposite of what it was before: rather than classifying everything as confirmed, now it is classifying everything as confirmed.

This is likely due to additional weight being placed on the unconfirmed light curves or possibly the dataset imbalance (although the imbalance is in favor of the confirmed curves, not the unconfirmed).

b) I went about remedying this error by further refining the class_weights section of the code, designed to prevent imbalance between the two types of light curves. However, because the algorithm classified everything as unconfirmed, I believe that the class_weights code was overcompensating for the greater number of confirmed light curves, resulting in a bias towards unconfirmed. Thus, I implemented the following, designed to prevent that new bias:

```
class_weights = [total / c for c in class_counts]
inv_freq = 1 / class_counts
norm_weights = inv_freq / inv_freq.sum()
class_weights_tensor = torch.tensor(norm_weights, dtype=torch.float32).to(device)
```

2. No. 2: Attempted with a smaller dataset, but one where there were 5 unconfirmed and 5 confirmed light curves (unlike in the trials(s) above) to increase the accuracy:
   a) Note: I placed the data I removed from the training and test sets into a "Held" folder for later use. I then set batch_size to 2, and added the following to print the class split after every few steps:

```
labels = [data.y.item() for data in graphs]
print("Label distribution (full dataset):", Counter(labels))
```

```
print("Train class distribution:", Counter(train_labels))
print("Val class distribution:", Counter(val_labels))
```

   b) Results:
      (1) I am currently working to process these at the moment (as of 6.3.25

C. Slideshow Preparation for Final Exam:
   1. This cycle, I also began work on my presentation for the final exam next week (6.11.25). The slideshow is shared and attached as a hyperlink here: LaMons.SRPII.Final.pptx

   2. Slideshow Order and Information:

a) Title: Utilizing Graph Neural Networks to Classify TESS Light Curves as a Novel Method for Confirming New Exoplanets

b) Introduction (include photograph of TESS):
   (1) TESS, the Transiting Exoplanet Survey Satellite, was launched in
   (2) NASA visualization of the Transit detection method (place in slides): https://svs.gsfc.nasa.gov/vis/a010000/a013000/a013022/Exoplanet_Single_Transit-HD_1080p.webm