body

June 15, 2024

To parents and supporters

J.S. DINGRA

Acknowledgment

I extend my heartfelt gratitude to several individuals whose support, guidance, and encouragement have been instrumental in the creation of this book on astronomy data analysis using Python, leveraging SDSS data.

Firstly, my deepest appreciation goes to my parents, Gurmeet Singh Dingra and Parminder Kaur Dingra, whose unwavering support and belief in my passion for astronomy have been my constant motivation.

A special note of thanks to my maternal uncle and aunt, Ravinderpal Singh and Rajinder Kaur, whose thoughtful gift of my first telescope ignited the spark for this celestial journey. Their encouragement has been an invaluable part of this venture.

I owe a debt of gratitude to Dr. Suprit Singh, an astrophysicist at IIT Delhi, whose guidance and continuous support have steered me at every turn, providing invaluable astrophysics and physics resources that enriched this work immeasurably.

I extend my thanks to Dr. Suprit Singh and Dr. Parvinder Singh for their meticulous review and feedback on this book, ensuring its accuracy and quality.

Special acknowledgment to Dragana Ilic and Isidora Jankov from the Department of Astronomy, University of Belgrade, Serbia, whose published work on AGN classification on GitHub significantly aided my understanding in this area.

Their expertise and contributions have been invaluable in shaping the content and understanding within these pages.

The data utilized in this book's analyses are sourced from the Sloan Digital Sky Survey (SDSS). We gratefully acknowledge the invaluable contribution of SDSS in advancing astronomical research and education.

This book stands as a testament to the invaluable contributions of these individuals and many others who have supported and inspired me on this astronomical journey.

Jashanpreet Singh Dingra

INDEX

I. Basic Python for Astronomy

1. Python Modules 5
2. NumPy 7
3. Pandas 13
4. Matplotlib 15

II. Astronomy Data Analysis

1. Astropy - Examples/Tasks 36
2. AGNs Classification - Final Project 67

Data and Material you may require while doing tasks: https://astrodingra.github.io/academics/material.html

PART I

Basic Python for Astronomy

1. PYTHON MODULES FOR ASTRONOMY

If you are interested in astronomy, you might have heard of Python - a popular programming language that is widely used in this field. Python's popularity stems from its ease of use, flexibility, and a vast ecosystem of modules that can be used to perform various tasks related to astronomy. Some of the most popular modules used in astronomy are Astropy, Astroquery, NumPy, Matplotlib, and Pandas. Each of these modules has its unique strengths and capabilities, which makes them essential tools for astronomers and researchers working in the field. We will be mostly using them throughout this book.

We will be using Jupyter notebook for this entire journey, so make sure that you have downloaded Jupyter notebook.

Astropy is an open-source and community-driven library for astronomers, which provides a set of tools for analyzing and manipulating astronomical data. Astropy is built on top of NumPy, which is another Python library used for scientific computing. Astropy is designed to be a one-stop-shop for astronomers, providing a wide range of functionalities such as handling coordinates, time, units, and physical constants. It also has a powerful framework for handling astronomical tables, which is essential for working with large astronomical datasets. One of the most significant advantages of Astropy is that it makes it easy to work with various file formats commonly used in astronomy. For instance, FITS (Flexible Image Transport System) files are a common data format used in astronomy, and Astropy provides a convenient way to read, write, and manipulate these files. It also provides various tools for image processing, such as aperture photometry and convolution.

Astroquery is a powerful Python package designed to streamline the process of querying astronomical databases. It offers a convenient interface for astronomers to access various online databases, retrieve data, and perform queries seamlessly within the Python environment. With Astroquery, astronomers can effortlessly access a wide range of astronomical data sources, from catalogs of stars and galaxies to observational data from telescopes and surveys. Its flexibility and ease of use make it an invaluable tool for astronomers and researchers working with large astronomical datasets.

In the realm of astronomy, accessing and analyzing data from large-scale surveys is fundamental. One such survey widely used by astronomers is the Sloan Digital Sky Survey (SDSS). Dealing with SDSS data often involves querying vast databases to retrieve information about celestial objects, including galaxies, stars, quasars, and more. Astroquery provides astronomers with a simplified approach to interact with SDSS data and perform queries efficiently. Its integration with SDSS and other major astronomical databases empowers researchers to explore and extract valuable insights from the vast repositories of astronomical data available.

We will learn SDSS data query in this book.

NumPy is another crucial module used in astronomy, providing powerful tools for performing mathematical operations on arrays and matrices. NumPy is a fundamental library for scientific computing and is widely used in various fields, including astronomy. NumPy's primary advantage is that it provides a fast and efficient way to perform mathematical operations on large datasets. It is also designed to work seamlessly with other Python libraries, making it a versatile tool for astronomy research.

Matplotlib is another critical library used in astronomy for data visualization. Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. It provides a wide range of plot types, including scatter plots, line plots, histograms, and 3D plots. Matplotlib is designed to be highly customizable, allowing astronomers to create publicationquality figures with ease. It also integrates well with other Python libraries, such as NumPy and Pandas, making it an essential tool for visualizing astronomical data.

Pandas is a Python library designed for data manipulation and analysis, and it is widely used in astronomy for the CSV data format. Pandas provides powerful tools for working with tabular data, including data cleaning, aggregation, and analysis. Pandas is built on top of NumPy, making it fast and efficient, even when working with large datasets. One of the significant advantages of Pandas is that it provides easy-to-use functions for handling missing data, making it a valuable tool for cleaning up messy astronomical datasets.

When working with astronomical data, it is essential to use the right tools and techniques to ensure accurate and reliable results. Python and its various modules have become a popular choice for astronomers due to their ease of use, flexibility, and wide range of functionalities. Whether you are working with astronomical tables, manipulating arrays, or creating visualizations, Python and its modules such as Astropy, NumPy, Matplotlib, and Pandas can help you achieve your goals. With these powerful tools at your disposal, you can explore the mysteries of the universe and uncover new insights into the workings of the cosmos.

[ ]:
```
#Must install these libraries before getting into real stuff
#Write the below code in jupyter *terminal* one by one
pip install numpy
pip install pandas
pip install matplotlib
pip install astropy
pip install astroquery
#this is the comment starting for #, to write your thoughts.
```

2. NUMPY

This chapter will cover NumPy in detail. NumPy (short for Numerical Python) provides an efficient interface to store and operate on dense data buffers. In some ways, NumPy arrays are like Python's built-in list type, but NumPy arrays provide much more efficient storage and data operations as the arrays grow larger in size. NumPy arrays form the core of nearly the entire ecosystem of data science tools in Python, so time spent learning to use NumPy effectively will be valuable no matter what aspect of astronomy interests you. Okay, let's begin by learning about NumPy. To get started, we need to import the NumPy module using the following Python code. Before that, you must have installed the NumPy module following the instructions provided in the first chapter.

[2]:
```
import numpy as np #np is the abbreviation that allows us to write 'np' instead␣
↪of writing 'numpy' in full within the code.
```

It is easy to write "np" instead of "numpy" in the code for making NumPy command. Throughout this book, we will import NumPy multiple times, so you must remember some shortcuts while writing the code.

> *For example, to display all the contents of the NumPy namespace, you can type this:*
> • *: np.<TAB>*
> *And to display NumPy's built-in documentation, you can use this:*
> • *: np?*
> *More detailed documentation, along with tutorials and other resources, can be found at http://www.numpy.org.*

A PYTHON LIST VERSUS A NUMPY ARRAY Let's first start by creating a Python list "L":

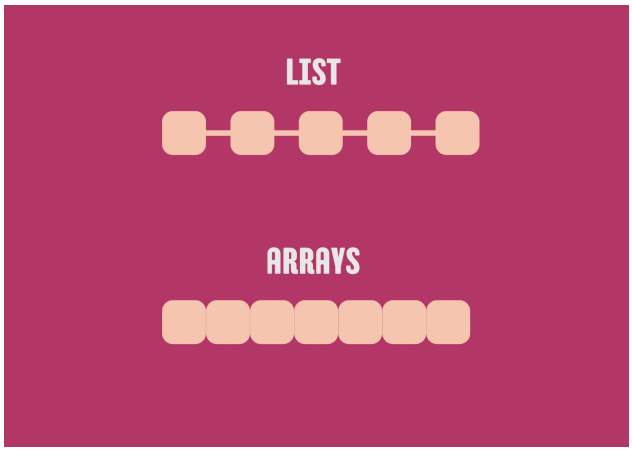
[2]:
```
L = list(range(10))
L
```

[2]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Note that 11.0 is a float, not an integer. Hence, anything written in decimal form is considered as a float. While using flexible lists in Python, each item in the list must contain its own type info and other information, which makes each item a complete Python object. This flexibility comes with a cost. However, if all variables are of the same type, it is more efficient to store data in a fixed-type array. This type of array has only one pointer to a contiguous block of data, while a Python list has a pointer to a block of pointers that each points to a full Python object. Even though the fixed-type array lacks the flexibility of the list, it is more efficient in storing and manipulating data. The difference between the two is illustrated in figure below.

The built-in array module can be used to create dense arrays of a uniform type:

[3]:
```
import array as ar
L2 = list(range(11))
A = ar.array('d' , L2) #'d' is the typecode which indicates that the output␣
↪will be in decimals
A
```



[3]:
```
array('d', [0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0])
```

If we want to clearly specify the data type of the resulting array, we can use the 'dtype' keyword:

[4]:
```
np.array([1,2,3,4,5], dtype="float32")
```

[4]:
```
array([1., 2., 3., 4., 5.], dtype=float32)
```

Ultimately, dissimilar to lists in Python, NumPy arrays can be multidimensional in a clear manner. Presented below is a method of initializing a multidimensional array using a list of lists.

[5]:
```
#a 3x5 matrix
np.array([range(i, i+5) for i in [2, 4, 6]])
```

[5]:
```
array([[ 2,  3,  4,  5,  6],
       [ 4,  5,  6,  7,  8],
       [ 6,  7,  8,  9, 10]])
```

Explaination of above code

for i in [2, 4, 6] iterates through the list [2, 4, 6]. range(i, i+5) creates a range of values starting from i up to i+5, excluding i+5. So, for i equal to 2, 4, and 6, it generates ranges [2, 3, 4, 5, 6], [4, 5, 6, 7, 8], and [6, 7, 8, 9, 10], respectively.

'linspace' command: used to generate random values in some range specified.

[6]:
```
# Creating an array with seven values evenly spaced between 1 and 2
np.linspace(1, 2, 7)
```

[6]:
```
array([1.        , 1.16666667, 1.33333333, 1.5       , 1.66666667,
       1.83333333, 2.        ])
```

[7]:
```
# you may also use the below syntax
# Below syntax means it will give values from 1 to 20, leaving the gap of 2,␣
 ↪which means odd numbers between 1 to 20
```

```
np.arange(1,20,2)
```

[7]:
```
array([ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19])
```

[8]:
```
# command for generating a random float 3x3 array
np.random.random((3, 3))
```

[8]:
```
array([[0.57797452, 0.86303127, 0.0768977 ],
       [0.28164383, 0.23494303, 0.7244336 ],
       [0.24751233, 0.25024219, 0.80961329]])
```

Do not try to give the integer command with the above command; otherwise, it will give a TypeError as shown below:

[9]:
```
np.random.random(2,5,(3,3))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[9], line 1
----> 1 np.random.random(2,5,(3,3))

File mtrand.pyx:438, in numpy.random.mtrand.RandomState.random()

TypeError: random() takes at most 1 positional argument (3 given)
```

Instead use the below syntax:

[10]:
```
np.random.randint(2,5,(3,3))
```

[10]:
```
array([[3, 4, 4],
       [2, 2, 4],
       [4, 4, 4]])
```

THE BASICS OF NUMPY ARRAYS In this section, we will showcase various examples that use NumPy array manipulation techniques to access data and subarrays, as well as to split, reshape, and join arrays. Although the operations demonstrated here may appear uninteresting and overly detailed, they form the fundamental components of many other examples highlighted in this book. Therefore, it is essential to familiarize yourself with them thoroughly We will only be covering the important array manipulations that are useful in astronomy.

Data Type	Description	Example
int8	8-bit integer	127
int16	16-bit integer	-32768
int32	32-bit integer	2147483647
int64	64-bit integer	-9223372036854775808
uint8	8-bit unsigned integer	255
uint16	16-bit unsigned integer	65535

Data Type	Description	Example
uint32	32-bit unsigned integer	4294967295
uint64	64-bit unsigned integer	18446744073709551615
float16	16-bit floating point	3.1415
float32	32-bit floating point	-2.71828
float64	64-bit floating point	1.41421356
complex64	64-bit complex	1 + 2j
complex128	128-bit complex	-3 - 4j

Indexing an Array: Getting the individual array element or, in layman's language, indexing of arrays can be thought of as demanding specific values from the array.

[11]:
```
L3 = list(range(10, 15))
A1 = ar.array('i', L3)
A1
```

[11]:
```
array('i', [10, 11, 12, 13, 14])
```

[12]:
```
# If we want to find the 1st value (or any value at a specific place) in the above array,
# we will use the following code
A1[0]  # This will give the value at place value = 1
```

[12]:
```
10
```

[13]:
```
A1[-1] # Negative sign will give the value from the end of the array
```

[13]:
```
14
```

[14]:
```
A1[-3]
```

[14]:
```
12
```

CONCLUSION: If we want the first value of the array, we need to input [0], but if we want the last value of the array, we need to input [-1].

[15]:
```
A2 = np.random.randint(5, 100, (3, 3))
A2
```

[15]:
```
array([[21, 11, 28],
       [95, 14, 97],
       [33, 15, 28]])
```

[16]:
```
A2[0, 0] # a11 term of matrix
```

[16]:
```
21
```

[17]:
```
A2[0, -1] # a13 term of matrix
```

[17]:
```
28
```

[18]:
```
A2[2, 1] # a31 term of matrix
```

[18]:
```
15
```

CONCLUSION: Row 1 is considered as 0 in Python. Similarly, the nth row is the (n+1)th row in Python (same with columns).

[19]:
```
# You can also modify values using any of the above index notations:
A2[0, 0] = 11
A2
```

[19]:
```
array([[11, 11, 28],
       [95, 14, 97],
       [33, 15, 28]])
```

Note that, unlike Python lists, NumPy arrays are fixed-type arrays. If you try to enter or modify the array with elements of a different datatype than that of the elements in the given array, it will give a TypeError.

[20]:
```
A1[0] = 6.78
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[20], line 1
----> 1 A1[0] = 6.78

TypeError: 'float' object cannot be interpreted as an integer
```

Array Slicing/Subarrays: Getting or separating subarrays from the array. Just as we can use square brackets to access individual array elements, we can also use them to access subarrays with the slice notation, marked by the colon (:) character. The NumPy slicing syntax follows that of the standard Python list; to access a slice of an array **x**, use this: x[start:stop:step] If any of these are unspecified, they default to the values start=0, stop=size of dimension, step=1. We'll take a look at accessing subarrays in one dimension and in multiple dimensions.

[21]:
```
A3 = np.arange(0 ,20)
A3[:11] #first 11 elements
```

[21]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

[22]:
```
A3[5::5] #starting from element 5 till the end, skipping 5 elements
```

[22]:
```
array([ 5, 10, 15])
```

[23]:
```
A3[6:7] #middle subarray
```

[23]:
```
array([6])
```

Similarly, here we can also use the negative sign to reverse the elements.

[24]:
```
A3[::-1] #all elements, reversed
```

[24]:
```
array([19, 18, 17, 16, 15, 14, 13, 12, 11, 10,  9,  8,  7,  6,  5,  4,  3,
        2,  1,  0])
```

[25]:
```
A3[6::-2] #reversed every other from index 6
```

[25]:
```
array([6, 4, 2, 0])
```

Multidimensional subarrays

[26]:
```
A4 = np.random.randint(5,10,(4 ,4))
A4
```

[26]:
```
array([[5, 6, 5, 6],
       [6, 5, 7, 9],
       [6, 8, 9, 6],
       [7, 9, 8, 9]])
```

[27]:
```
A4 [:3,:2] #three rows, two columns
```

[27]:
```
array([[5, 6],
       [6, 5],
       [6, 8]])
```

Reversing a multidimensional subarray:

[28]:
```
A4[::-1,::-1]
```

[28]:
```
array([[9, 8, 9, 7],
       [6, 9, 8, 6],
       [9, 7, 5, 6],
       [6, 5, 6, 5]])
```

You can refer to the NumPy Documentation to learn more about the advantages of NumPy arrays.

3. PANDAS

In this chapter, we will be learning about data frames in pandas. However, most of the data in astronomy is in the form of FITS files, but sometimes you may encounter data in the form of CSV (comma-separated values) format. This chapter will be shorter compared to the rest of the chapters since we deal more frequently with FITS files. In astronomy, we often work with large datasets from various sources. In this chapter, we will explore how to load and clean data in pandas. We will cover techniques for handling missing data, removing duplicates, and converting data types. However, we will learn to plot the data in Chapter 4 Let's first create a CSV file with empty and duplicate values

[29]:
```
import csv

# Define data to write to the CSV file
data = [
    ['Galaxy', 'Redshift'],
    ['NGC 1068', '0.003793'],
    ['M31', '0.000282'],
    ['M33', '0.000921'],
    ['M51', '0.000801'],
    ['M81', '0.000680'],
    ['M82', '0.000677'],
    ['NGC 253', '0.000823'],
    ['NGC 300', '0.000811'],
    ['NGC 891', '0.002592'],
    ['NGC 1365', '0.005500'],
    ['NGC 1566', '0.005473'],
    ['NGC 1672', '0.004767'],
    ['NGC 2442', '0.003500'],
    ['NGC 2903', '0.002245'],
    ['NGC 2997', '0.002025'],
    ['NGC 3031', '0.000804'],
    ['NGC 3627', '0.002346'],
    ['NGC 4258', '0.001548'],
    ['NGC 4414', '0.001552'],
    ['NGC 4559', '0.001776'],
    ['NGC 4565', '0.002528'],
    ['NGC 4725', '0.004092'],
    ['NGC 5194', '0.001497'],
    ['NGC 5457', '0.000674'],
    ['NGC 5907', '0.002224'],
    ['NGC 6946', '0.000927'],
    ['NGC 7331', '0.002045'],
    ['UGC 1810', ''],
    ['UGC 1922', '0.003500'],
    ['UGC 2855', '0.004600'],
    ['', ''],
]
```

```
# Write data to the CSV file
with open('galaxies.csv', mode='w', newline='') as file:
    writer = csv.writer(file)
    writer.writerows(data)
```

Now, after running this code, you will see the CSV file in the same folder as your Python file. Let's load our CSV file.

[30]:
```
import pandas as pd
df = pd.read_csv("galaxies.csv")
df.head()
```

[30]:

	Galaxy	Redshift
0	NGC 1068	0.003793
1	M31	0.000282
2	M33	0.000921
3	M51	0.000801
4	M81	0.000680

Removing duplicate and empty entries

[31]:
```
import numpy as np

# Drop rows with missing values
df.dropna(inplace=True)

# Drop duplicates
df.drop_duplicates(inplace=True)

# Replace missing values
df['Redshift'].fillna(value=np.mean(df['Redshift']), inplace=True)

# Remove outliers
df = df[np.abs(df['Redshift'] - df['Redshift'].mean()) <= (3 * df['Redshift'].
 ↪std())]

# Rename columns
df.rename(columns={'Redshift': 'new_Redshift'}, inplace=True)

# Export cleaned data to a new file
df.to_csv('cleaned_data_galaxies.csv', index=False)
```

Oh yay! We have successfully cleaned the data. Our new data is ready to be analyzed in Chapter 4.

4. MATPLOTLIB

In astronomy, data analysis is a crucial step in extracting scientific insights from astronomical observations. One of the most important aspects of data analysis is visualization, which allows astronomers to explore their data, identify patterns and trends, and communicate their findings effectively. Matplotlib is a popular Python library for creating high-quality, customizable visualizations of scientific data, including astronomical data.

In this chapter, we will introduce Matplotlib and its key features for creating visualizations of astronomical data. We will cover the basics of Matplotlib, including how to create simple line plots and scatter plots. We will also explore more advanced features of Matplotlib, such as customizing plot styles, adding labels and annotations, and creating multi-panel plots

Section 1: Getting Started with Matplotlib

- Importing Matplotlib in our Python notebook
- Basic concepts of Matplotlib: Figure, Axes, and Plots
- Creating a simple line plot with Matplotlib
- Customizing plot styles: line color, style, and width
- Adding axis labels and a title to a plot
- Saving plots to a file

Section 2: Exploring Astronomical Data with Matplotlib

- Loading astronomical data into Python with astropy or other libraries
- Creating scatter plots and histogram plots of astronomical data
- Adding error bars to scatter plots
- Customizing plot markers and colors
- Creating density plots and contour plots for 2D data

Section 3: Advanced Visualization Techniques with Matplotlib

- Creating multi-panel plots with subplots and gridspec
- Customizing plot layouts and aspect ratios
- Adding annotations and text to plots
- Creating color maps for visualizing data with color
- Visualizing astronomical images with imshow and wcsaxes

The last two sections will be part of Chapter 5

By the end of this chapter (Chapters 4 and 5), you will have a solid understanding of Matplotlib and its capabilities for visualizing astronomical data. You will be able to create a variety of plot types and customize them to communicate your findings effectively. Additionally, you will be familiar

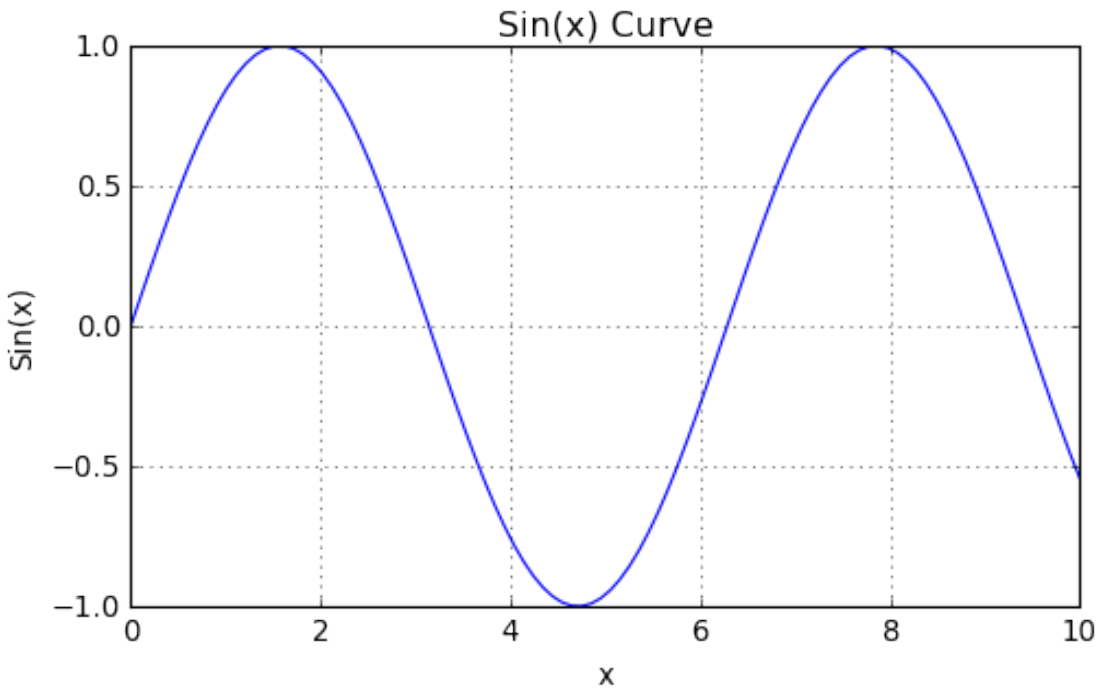
with best practices for using Matplotlib in astronomy data analysis and be able to apply these techniques to your own research projects.

GETTING STARTED WITH MATPLOTLIB

Numpy Based: Let's Create our Very First Plot

Importing the libraries before writing the code is must to run the code.

[71]:
```
# Importing necessary libraries: Matplotlib for plotting, NumPy for numerical␣
 ↪operations
import matplotlib.pyplot as plt
import numpy as np

# Generating values for x-axis using NumPy's linspace function (from 0 to 10␣
 ↪with 100 points)
x = np.linspace(0, 10, 100)

# Computing corresponding y values for the sin(x) curve
y = np.sin(x)

plt.figure(figsize=(7,4)) # Width = 7 inches and Height = 4 inches

plt.gcf().set_facecolor('none')  # Set the figure's face color to none␣
 ↪(transparent)

# Setting up the plot
plt.grid()                      # Displaying a grid in the plot
plt.xlabel('x')                 # Labeling x-axis as 'x'
plt.ylabel('Sin(x)')            # Labeling y-axis as 'Sin(x)'
plt.title('Sin(x) Curve')       # Setting the title of the plot as 'Sin(x)␣
 ↪Curve'

# Plotting the sin(x) curve with x values on the x-axis and y values on the␣
 ↪y-axis
plt.plot(x, y)

# Using a classic style for the plot for a traditional look and feel
plt.style.use('classic')

# Saving the plot as a PNG image file with high resolution (DPI = 1000)
plt.savefig('sinx.png', dpi=1000)

# Displaying the plot
plt.show()
```



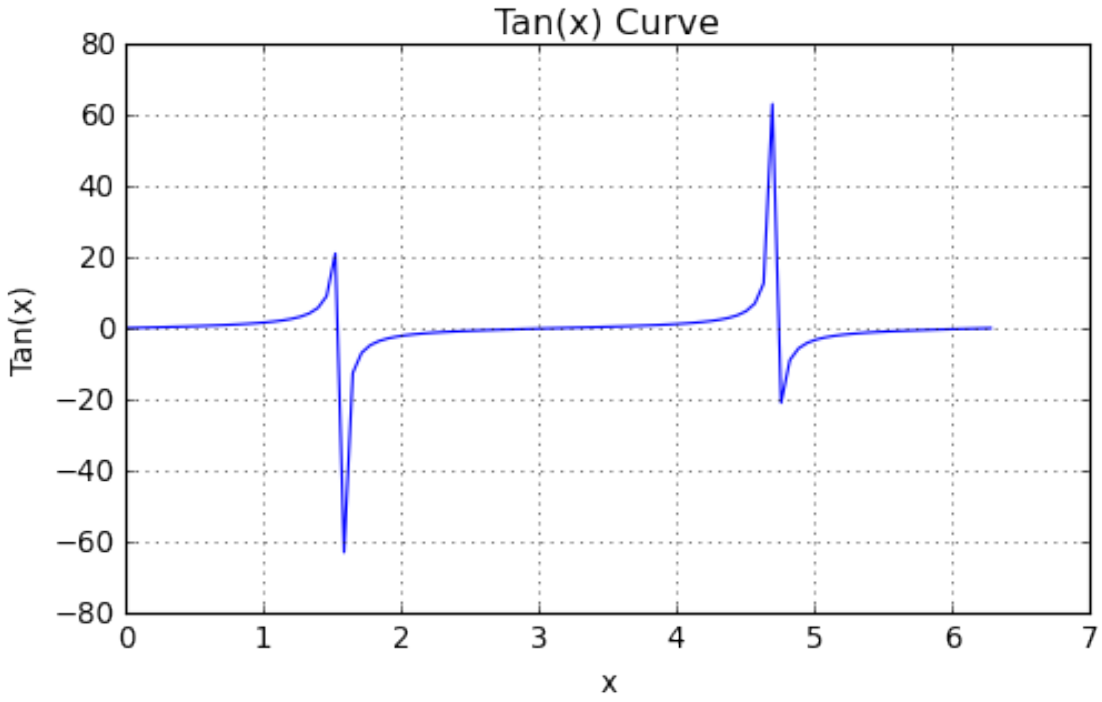
TASK 1: Make cosine graph using numpy and matplotlib.

TASK 1 SOLUTION

Similarly, we can make a tan(x) curve too:

[46]:
```
# Generating x values from 0 to 2 with 100 points using NumPy's linspace
x = np.linspace(0, 2 * np.pi, 100)

# Calculating corresponding y values for the tan(x) function
y = np.tan(x)

plt.figure(figsize=(7,4)) # Width = 7 inches and Height = 4 inches

plt.gcf().set_facecolor('none')  # Set the figure's face color to none␣
 ↪(transparent)

# Setting x-axis label as 'x'
plt.xlabel('x')

# Setting y-axis label as 'Tan(x)'
plt.ylabel('Tan(x)')

# Setting title of the plot as 'Tan(x) Curve'
plt.title('Tan(x) Curve')
```

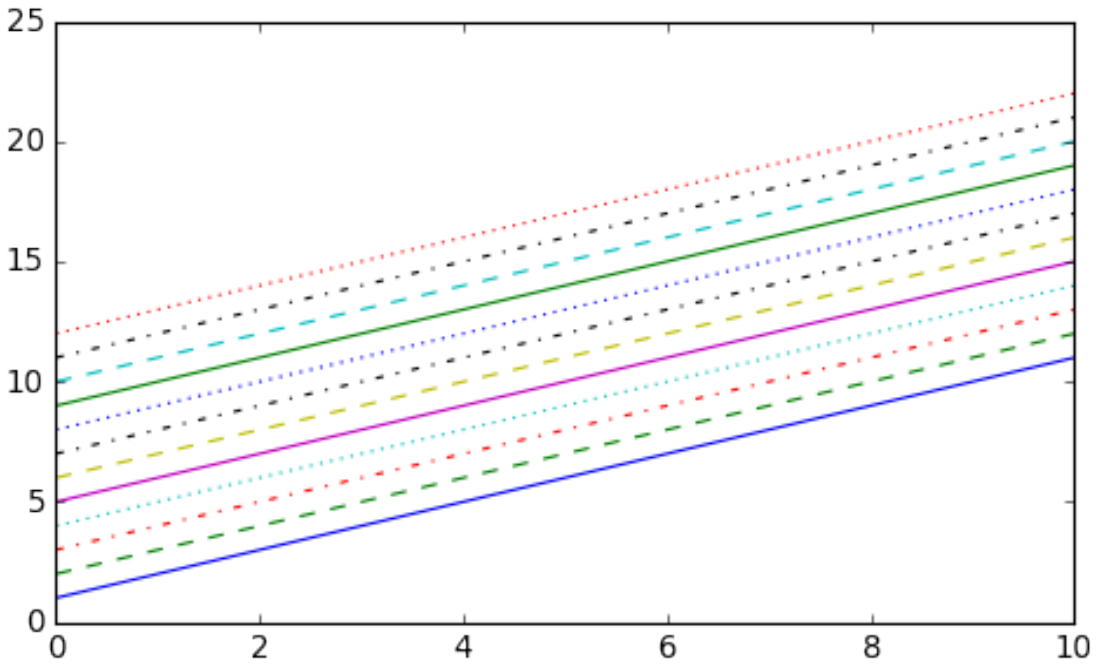
```
# Displaying a grid on the plot
plt.grid()

# Plotting the tan(x) curve with x values on the x-axis and y values on the␣
 ↪y-axis
plt.plot(x, y)

# Using a classic style for the plot for a traditional look and feel
plt.style.use('classic')

# Saving the plot as a PNG image file with high resolution (DPI = 1000)
plt.savefig('tanx.png', dpi=1000)

# Displaying the plot
plt.show()
```



Fancy designs:

[77]:
```
plt.figure(figsize=(7,4))
# Plotting lines with different line styles
plt.plot(x, x + 1, linestyle='solid')    # Solid line style
plt.plot(x, x + 2, linestyle='dashed')   # Dashed line style
plt.plot(x, x + 3, linestyle='dashdot')  # Dash-dot line style
```

```
plt.plot(x, x + 4, linestyle='dotted')   # Dotted line style

# Alternatively, using shorthand codes:
plt.plot(x, x + 5, linestyle='-')    # Solid line style
plt.plot(x, x + 6, linestyle='--')   # Dashed line style
plt.plot(x, x + 7, linestyle='-.')   # Dash-dot line style
plt.plot(x, x + 8, linestyle=':')    # Dotted line style

# Plotting lines with different line styles and colors
plt.plot(x, x + 9, '-g', label='Solid Green')      # Solid line style, green␣
 ↪color
plt.plot(x, x + 10, '--c', label='Dashed Cyan')     # Dashed line style, cyan␣
 ↪color
plt.plot(x, x + 11, '-.k', label='Dash-dot Black')  # Dash-dot line style,␣
 ↪black color
plt.plot(x, x + 12, ':r', label='Dotted Red')       # Dotted line style, red␣
 ↪color

plt.gcf().set_facecolor('none')
plt.show()
```



Displaying your plots in Matplotlib is essential to interpreting and utilizing your visualizations effectively. However, how you choose to view your Matplotlib plots depends on the context in which you're working.

There are three primary contexts for using Matplotlib:

1. **Script:** If you're writing code in a Python script, you'll typically include `plt.show()` at the end of your script. This command prompts the plot to appear in a separate window.
2. **IPython Terminal:** When using Matplotlib in an IPython terminal, calling `plt.show()` will also display the plot. However, in some cases, the terminal configuration might automatically show the plot without needing this command.
3. **IPython Notebook:** In an IPython notebook, plots will automatically appear inline without explicitly calling `plt.show()`. This behavior is due to the notebook's functionality to display the plots directly below the code cell where they are generated.

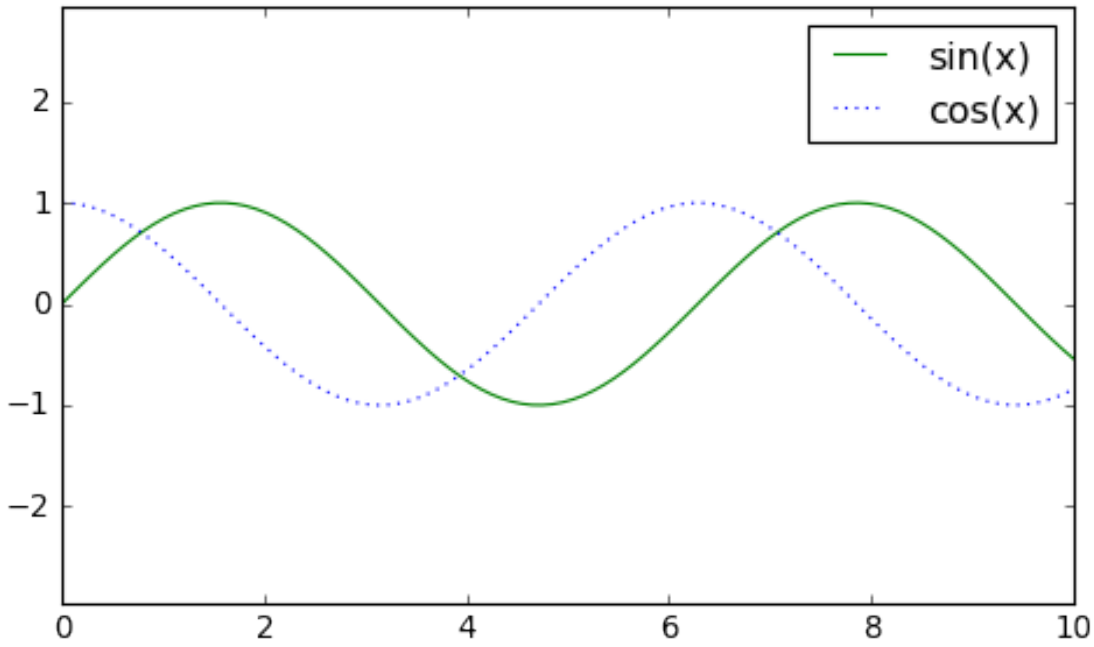
Understanding these different contexts helps in deciding when and how to use `plt.show()` to visualize your Matplotlib plots effectively.

Let's improve our plots:

• When multiple lines are being shown within a single axes, it can be useful to create a plot legend that labels each line type. Again, Matplotlib has a built-in way of quickly creating such a legend. It is done via the `plt.legend()` method.

• The legend displays labels for different elements of the plot, such as lines, markers, or other graphical elements.

[44]:
```
# Generating x values from 0 to 10 with 100 points using NumPy's linspace
x = np.linspace(0, 10, 100)

# Set a figure size
plt.figure(figsize=(7,4)) # Width = 7 inches and Height = 4 inches

plt.gcf().set_facecolor('none')  # Set the figure's face color to none␣
 ↪(transparent)

# Plotting the sin(x) curve in green color with solid line style and labeling it
plt.plot(x, np.sin(x), '-g', label='sin(x)')

# Plotting the cos(x) curve in blue color with dotted line style and labeling it
plt.plot(x, np.cos(x), ':b', label='cos(x)')

# Making the x-axis and y-axis have equal scaling
plt.axis('equal')

# Displaying a legend based on labeled lines
plt.legend()

# Saving the combined plot as a PNG image file with high resolution (DPI = 1000)
plt.savefig('combinedx.png', dpi=1000)
```



You can save your plot using the following code:

[ ]:
```
# Saving the plot as a PNG image file with high resolution (DPI = 1000)
# DPI (dots per inch) refers to the quality of the image
plt.savefig('my_plot.png', dpi=1000)

# Saving the plot as a PDF file
plt.savefig('my_plot.pdf', format='pdf')

# Additional options for saving the figure:

# Specifying the bounding box in inches [left, bottom, width, height]
plt.savefig('my_plot.pdf', format='pdf', bbox_inches='tight')

# Setting the figure's transparent background (useful for plots with␣
 ↪transparency)
plt.savefig('my_plot.png', dpi=1000, transparent=True)

# Adjusting the quality when saving as JPG with a specified quality level␣
 ↪(0-100)
plt.savefig('my_plot.jpg', format='jpg', quality=95)

# Saving with a specific aspect ratio
plt.savefig('my_plot.png', dpi=1000, aspect='auto')
```

```
# Setting the face color of the saved figure or simple background
plt.savefig('my_plot.png', dpi=1000, facecolor='lightblue')

# Adding metadata (like title, author, etc.) to the saved file
plt.savefig('my_plot.png', dpi=1000, metadata={'Title': 'My Plot', 'Author':␣
 ↪'Author Name'})
```

Below are the file types in which you can save your plots:

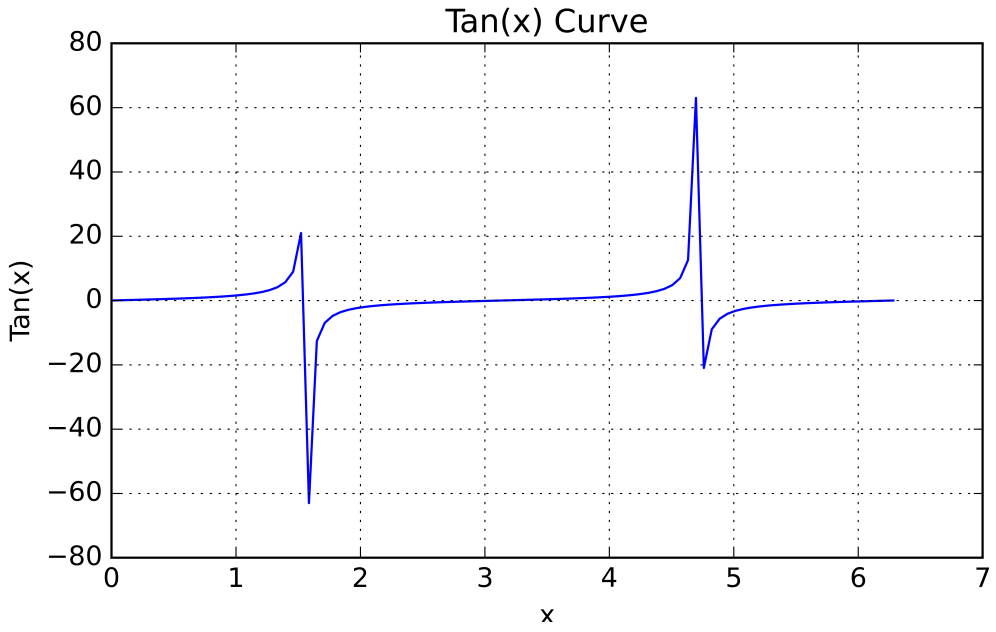
[35]:
```
# Get supported file types for saving figures
supported_filetypes = fig.canvas.get_supported_filetypes()

# Print the supported file types
print("Supported file types for saving figures:", supported_filetypes)
```

```
Supported file types for saving figures: {'eps': 'Encapsulated Postscript',
'jpg': 'Joint Photographic Experts Group', 'jpeg': 'Joint Photographic Experts
Group', 'pdf': 'Portable Document Format', 'pgf': 'PGF code for LaTeX', 'png':
'Portable Network Graphics', 'ps': 'Postscript', 'raw': 'Raw RGBA bitmap',
'rgba': 'Raw RGBA bitmap', 'svg': 'Scalable Vector Graphics', 'svgz': 'Scalable
Vector Graphics', 'tif': 'Tagged Image File Format', 'tiff': 'Tagged Image File
Format', 'webp': 'WebP Image Format'}
```

Displaying saved images using IPython

[49]:
```
from IPython.display import Image

# Display the image 'tanx.png'
Image('tanx.png')
```

[49]:

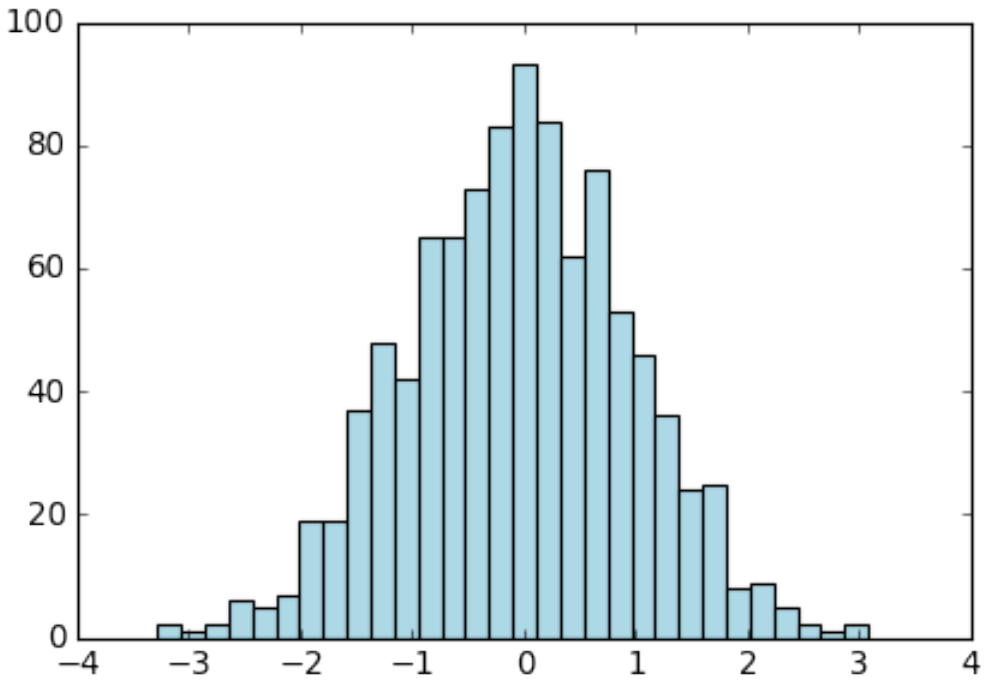


You may also draw histograms, scatter plots, and piechart using the following code: (however this an extra information, just for learning purpose)

HISTOGRAM

[42]:
```
# Generating sample data
data = np.random.normal(0, 1, 1000)  # Example data for the histogram

# Set the figure size
plt.figure(figsize=(6, 4))  # Width: 6 inches, Height: 4 inches

plt.gcf().set_facecolor('none')  # Set the figure's face color to none␣
 ↪(transparent)
# Create a histogram
plt.hist(data, bins=30, color='lightblue')
# Bin is the number of bar to be taken in the plot of distribution
# You may increase or decrease the value of bin to see what changes
```

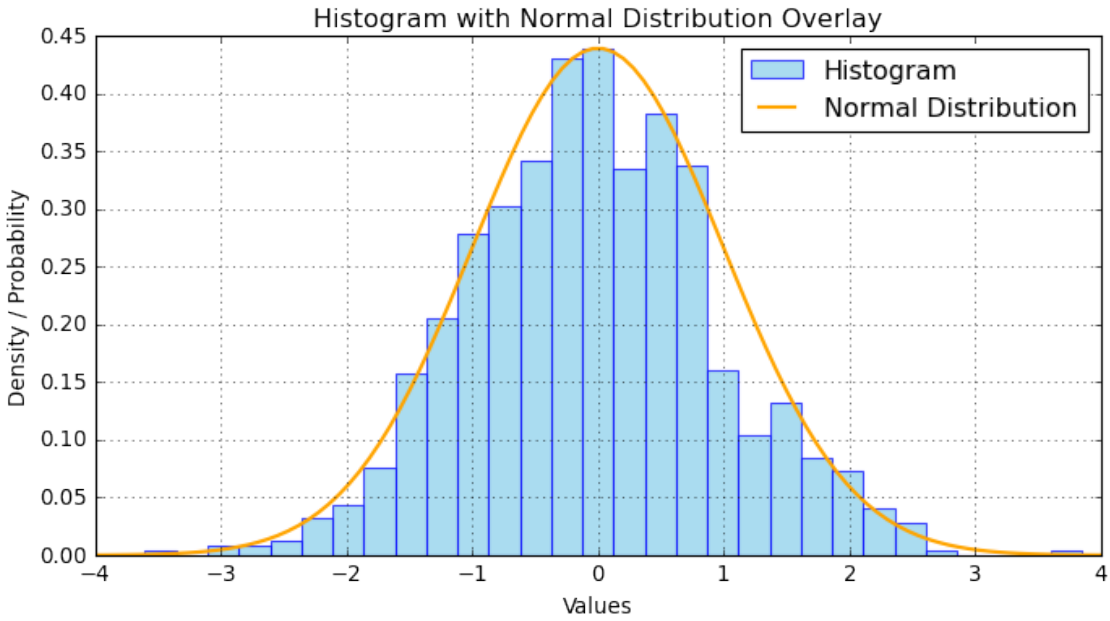
[42]:
```
(array([ 2.,  1.,  2.,  6.,  5.,  7., 19., 19., 37., 48., 42., 65., 65.,
        73., 83., 93., 84., 62., 76., 53., 46., 36., 24., 25.,  8.,  9.,
         5.,  2.,  1.,  2.]),
 array([-3.28197494, -3.06996045, -2.85794596, -2.64593147, -2.43391698,
        -2.22190249, -2.009888  , -1.79787351, -1.58585902, -1.37384453,
        -1.16183004, -0.94981555, -0.73780106, -0.52578657, -0.31377208,
        -0.10175759,  0.1102569 ,  0.32227139,  0.53428588,  0.74630037,
         0.95831486,  1.17032935,  1.38234384,  1.59435833,  1.80637282,
         2.01838731,  2.2304018 ,  2.44241629,  2.65443078,  2.86644527,
         3.07845976]),
 <BarContainer object of 30 artists>)
```



Histogram with Line Plot Overlay* :Every line of code is important to note

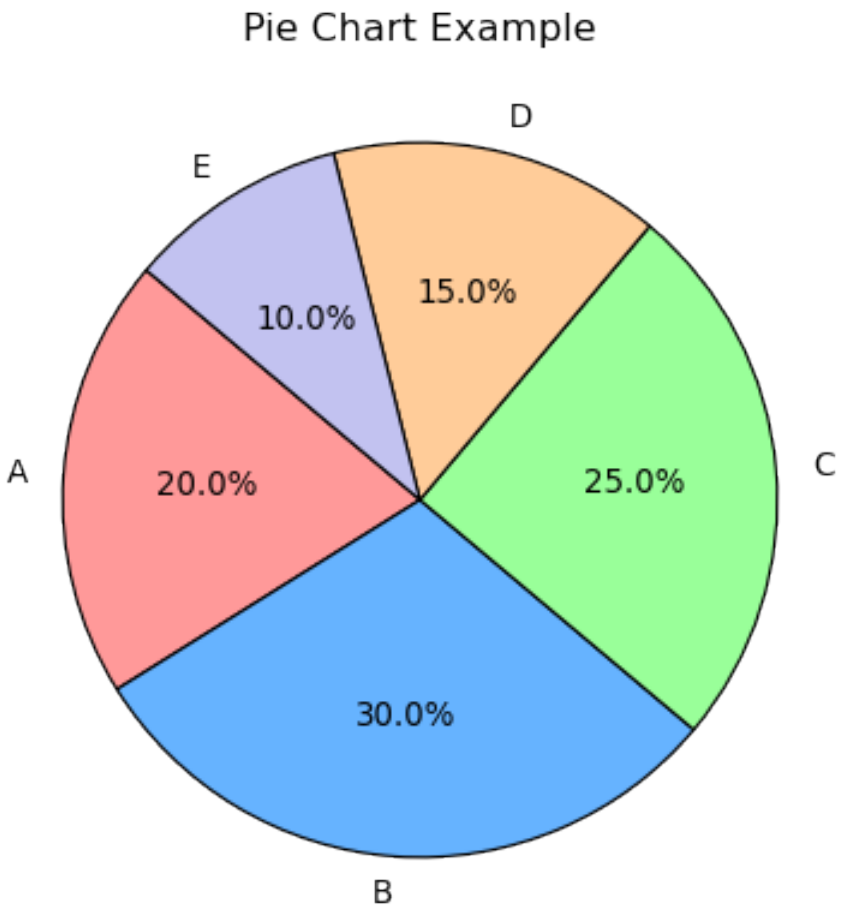
[31]:
```
# Importing libraries is must (once)
import numpy as np
import matplotlib.pyplot as plt

# Generating sample data (1000 data points) from a normal distribution with␣
 ↪mean 0 and standard deviation 1
data = np.random.normal(0, 1, 1000)

# Set the figure size
plt.figure(figsize=(10, 5))  # Width: 10 inches, Height: 5 inches

# Create a histogram of the sample data
plt.hist(data, bins=30, color='skyblue', edgecolor='blue', alpha=0.7,␣
 ↪label='Histogram', density=True)
# 'bins=30' sets the number of bins in the histogram
# 'color' sets the color of the bars in the histogram
# 'edgecolor' sets the color of the edges of the bars
# 'alpha' sets the transparency of the bars
# 'label' provides a label for the histogram bars
# 'density=True' normalizes the histogram so that the total area under the bars␣
 ↪sums to 1 (represents a probability density)
```

```
# Calculate values for a normal distribution line plot
mu, sigma = 0, 1  # mean and standard deviation of the normal distribution
x = np.linspace(-4, 4, 100)  # Generating 100 points between -4 and 4 for x-axis

# Calculate the probability density function (PDF) of the normal distribution
normal_dist = (1 / (np.sqrt(2 * np.pi) * sigma)) * np.exp(-(x - mu) ** 2 / (2 *␣
 ↪sigma ** 2))

# Scale the normal distribution to fit the histogram by adjusting its maximum␣
 ↪to match the histogram's maximum
bin_edges, _ = np.histogram(data, bins=30, density=True)
scale_factor = bin_edges.max() / normal_dist.max()
normal_dist *= scale_factor

# Overlay a line plot (normal distribution) on top of the histogram
plt.plot(x, normal_dist, color='orange', linewidth=2, label='Normal␣
 ↪Distribution')
# 'color' sets the color of the line plot
# 'linewidth' sets the width of the line
# 'label' provides a label for the line plot

# Set labels and title for the plot
plt.title('Histogram with Normal Distribution Overlay')
plt.xlabel('Values')
plt.ylabel('Density / Probability')
plt.legend()  # Display the legend showing labels for the histogram and line␣
 ↪plot

# Display the plot
plt.grid(True)  # Show gridlines on the plot
plt.gcf().set_facecolor('none')  # Set the figure's face color to none␣
 ↪(transparent)
#Remove above line to see whats the change. The background of the plot will␣
 ↪become grey
plt.show()# Show the plot
```



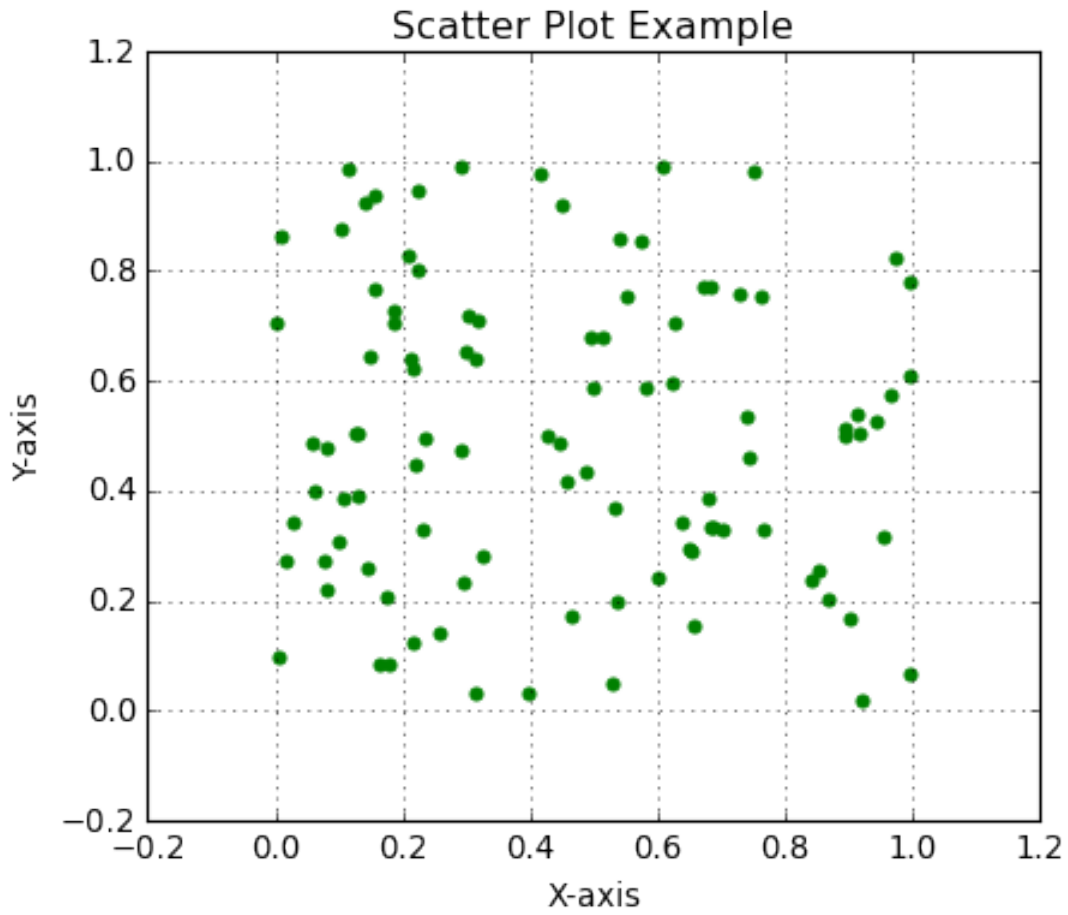
PIE CHART

[19]:
```
# Generating sample data
x = [20, 30, 25, 15, 10]  # Example data for the pie chart
labels = ['A', 'B', 'C', 'D', 'E']  # Labels for each wedge in the pie chart

# Define colors for the wedges (you can specify colors as hex values, names, or␣
 ↪RGB tuples)
colors = ['#ff9999', '#66b3ff', '#99ff99', '#ffcc99', '#c2c2f0']  # Just to␣
 ↪make fancy
plt.figure(figsize=(6, 6))  # Width: 6 inches, Height: 6 inches

plt.gcf().set_facecolor('none')  # Set the figure's face color to none␣
 ↪(transparent)

# Create a pie chart using 'x' values and defined labels, with custom colors
plt.pie(x, labels=labels, autopct='%1.1f%%', startangle=140, colors=colors)
plt.title('Pie Chart Example')

plt.show()
```



SCATTER PLOT

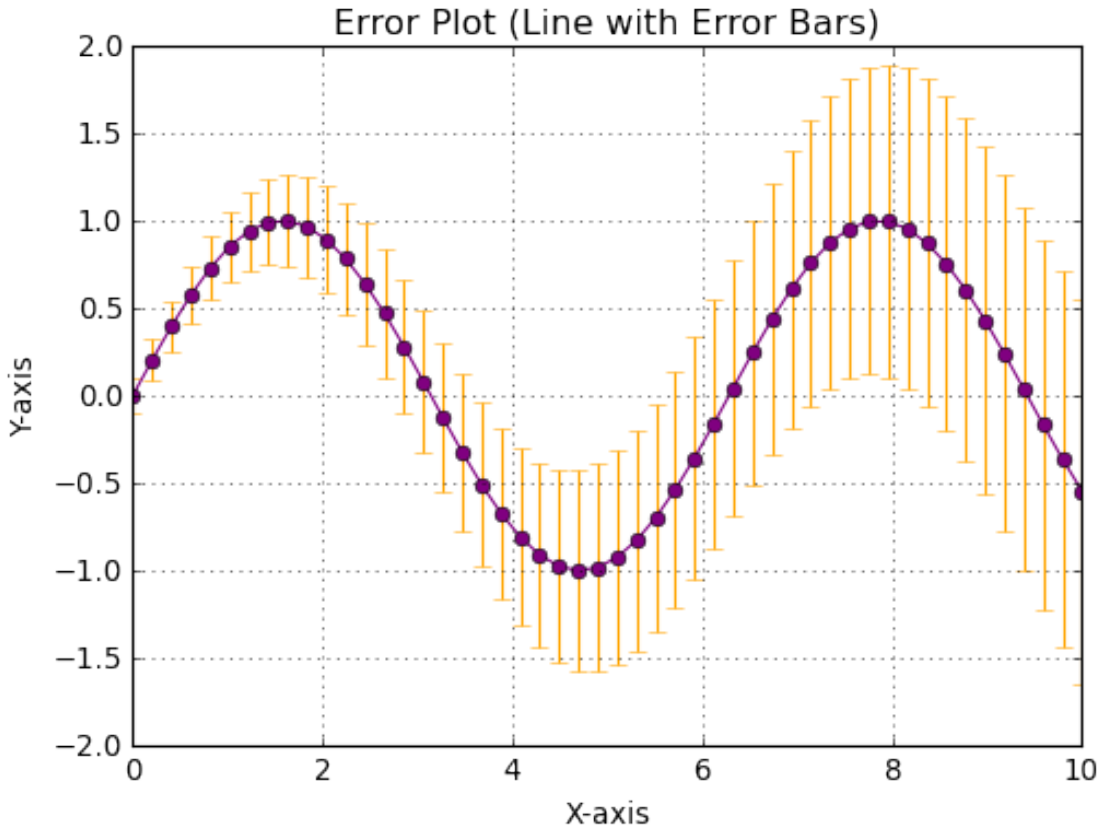
[36]:
```
# Generating sample data
x = np.random.rand(100)
y = np.random.rand(100)  # Example data for the scatter plot
plt.figure(figsize=(7,5)) # Width= 7 inches, Height= 5 inches

# Create a scatter plot
plt.scatter(x, y, color='green', marker='o')
plt.title('Scatter Plot Example')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.grid(True)
plt.gcf().set_facecolor('none')  # Set the figure's face color to none␣
↪(transparent)
plt.show()
```



ERROR PLOT

[39]:
```
# Generating sample data
x = np.linspace(0, 10, 50)
y = np.sin(x)
error = 0.1 + 0.1 * x  # Example error values (could be standard deviation,␣
 ↪SEM, etc.)
plt.figure(figsize=(7,5))# Width= 7 inches, Height= 5 inches

# Create an error plot
plt.errorbar(x, y, yerr=error, fmt='-o', color='purple', ecolor='orange',␣
 ↪elinewidth=1, capsize=4)
plt.title('Error Plot (Line with Error Bars)')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.grid(True)
plt.gcf().set_facecolor('none')  # Set the figure's face color to none␣
 ↪(transparent)
```

```
plt.show()
```

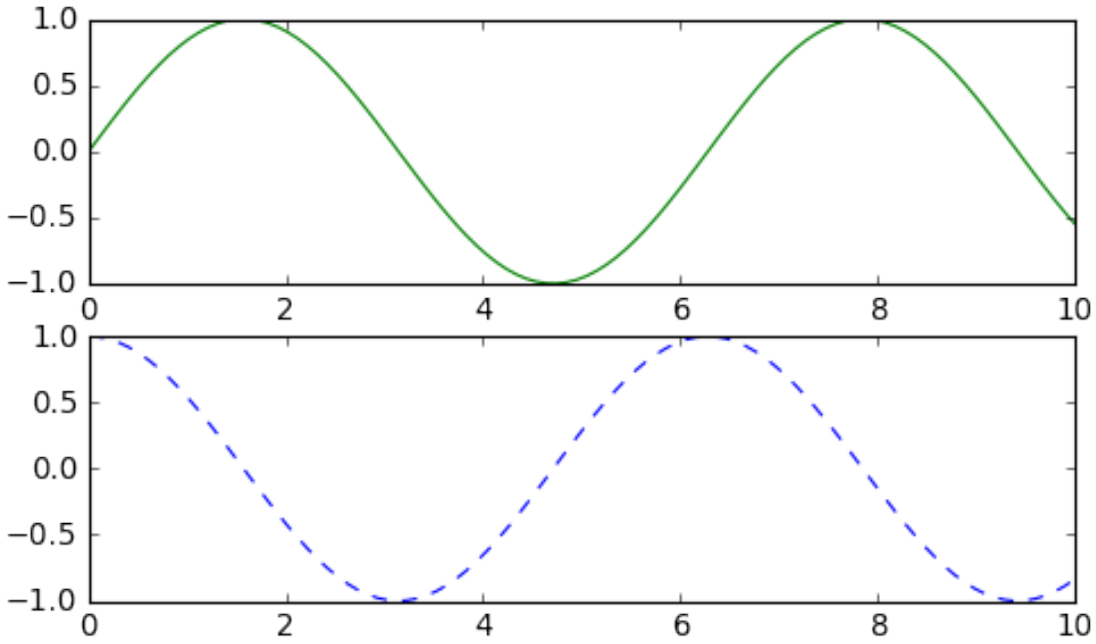


Making Subplots In astronomy, subplots are very useful to compare two plots for specific data. There are two ways to make subplots one using `plt.subplot(x,y,z)` where x,y,z are rows, columns, panel number respectively. other using `fig, axs = plt.subplots(p,q)` where p,q are order of matrix formed by subplot.

1. Using plt.subplot():

[51]:
```
# Create a plot figure
plt.figure(figsize=(7,4)) # Width = 7 inches and Height = 4 inches

plt.gcf().set_facecolor('none')  # Set the figure's face color to none␣
↪(transparent)
# Save the figure with high dpi
plt.savefig('subplot_cos_sin_x.png', dpi=2000)

# Create the first of two panels and set the current axis
plt.subplot(2, 1, 1)  # (rows, columns, panel number)
```

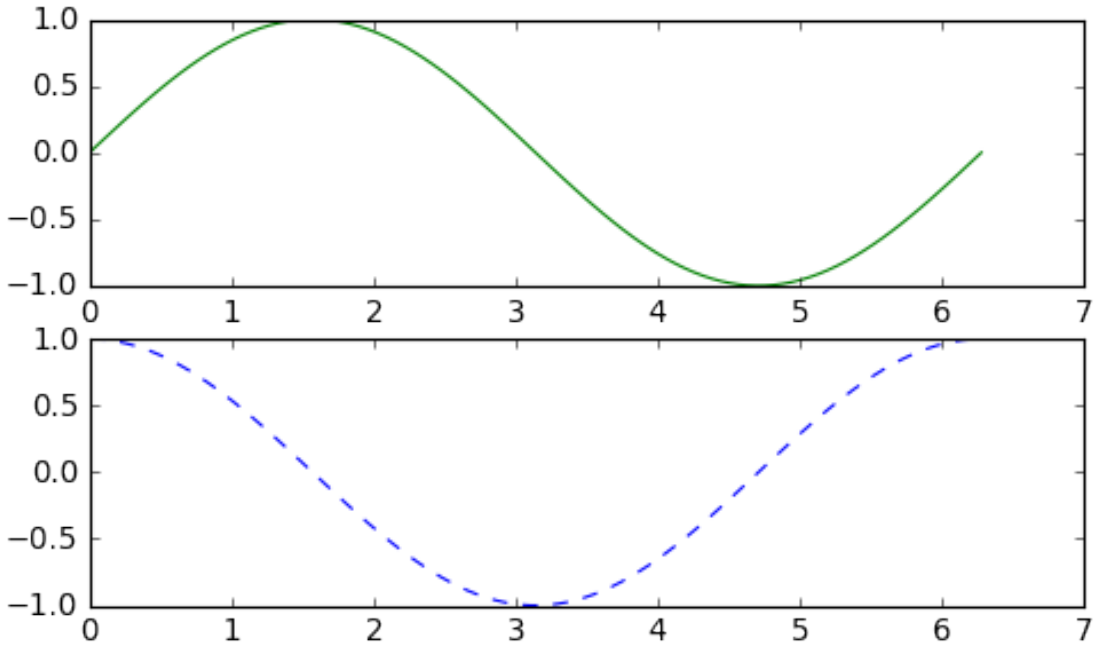
```
# Plot the sine function
plt.plot(x, np.sin(x), '-g')  # Plot the sine function in green

# Create the second panel and set the current axis
plt.subplot(2, 1, 2)

# Plot the cosine function
plt.plot(x, np.cos(x), '--b')  # Plot the cosine function in blue

plt.show()
```



2. Using plt.subplot() (Object-oriented approach):

[63]:
```
# Generate data
x = np.linspace(0, 2*np.pi, 100)
y1 = np.sin(x)
y2 = np.cos(x)

# Using plt.subplots() - Object-oriented approach
fig, axs = plt.subplots(2, 1)  # Create a figure with 2 subplots arranged in a␣
↪2x1 grid

fig.set_size_inches(7, 4)  # Resize the figure (width=8, height=6)

axs[0].plot(x, y1, '-g')  # Plot data on the first subplot
```

```
axs[1].plot(x, y2, '--b')  # Plot data on the second subplot

fig.patch.set_alpha(0)  # Set figure background transparency to 0.5 (adjust as␣
 ↪needed)

plt.show()
```

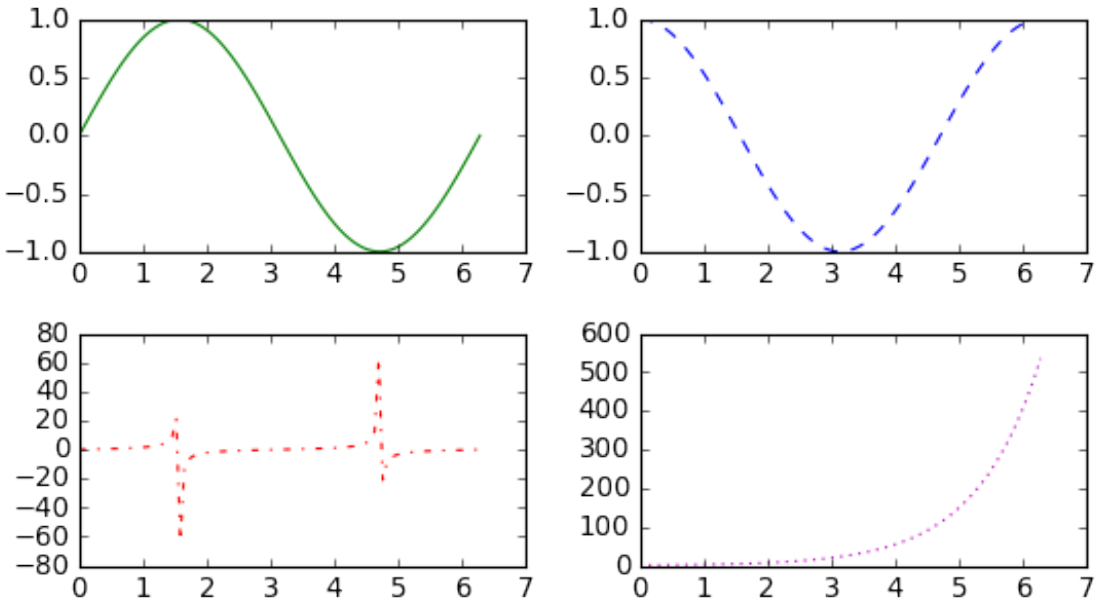


One more example:

Method 1:

[69]:
```
# Generate data
x = np.linspace(0, 2*np.pi, 100)
y1 = np.sin(x)
y2 = np.cos(x)
y3 = np.tan(x)
y4 = np.exp(x)

plt.figure(figsize=(7,4))

# Using plt.subplot()
plt.subplot(2, 2, 1)  # Create the first subplot in a 2x2 grid
plt.plot(x, y1, '-g')  # Plot data on the first subplot

plt.subplot(2, 2, 2)  # Create the second subplot in a 2x2 grid
plt.plot(x, y2, '--b')  # Plot data on the second subplot
```

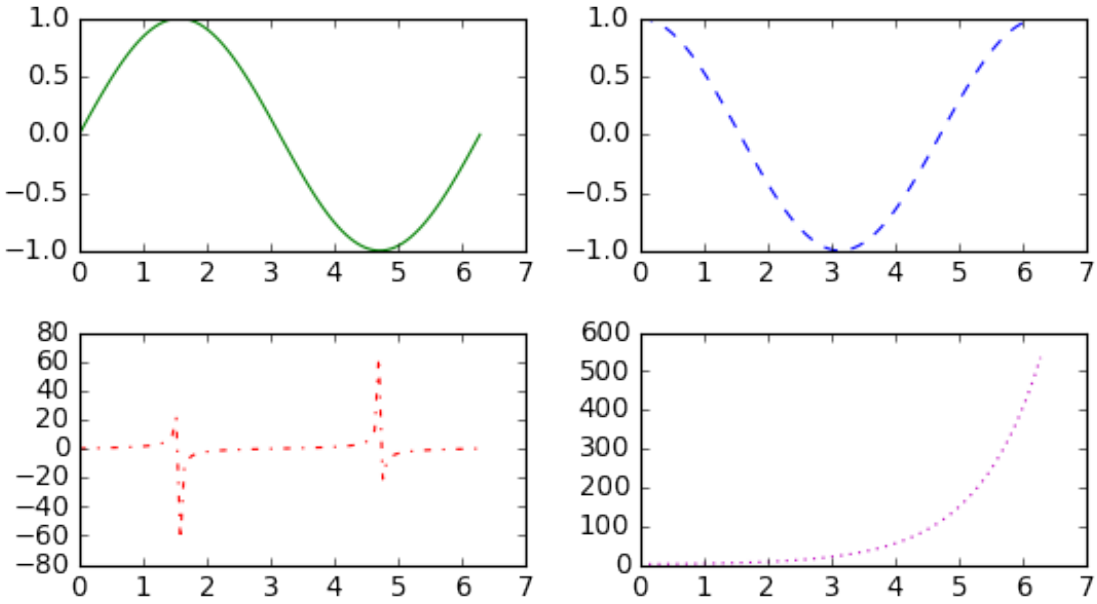
```
plt.subplot(2, 2, 3)  # Create the third subplot in a 2x2 grid
plt.plot(x, y3, '-.r')  # Plot data on the third subplot

plt.subplot(2, 2, 4)  # Create the fourth subplot in a 2x2 grid
plt.plot(x, y4, ':m')  # Plot data on the fourth subplot

plt.gcf().set_facecolor('none')

plt.tight_layout()  # Adjust spacing between subplots
plt.show()
```



Method 2:

[70]:
```
import matplotlib.pyplot as plt
import numpy as np

# Generate data
x = np.linspace(0, 2*np.pi, 100)
y1 = np.sin(x)
y2 = np.cos(x)
y3 = np.tan(x)
y4 = np.exp(x)

# Using plt.subplots() - Object-oriented approach
fig, axs = plt.subplots(2, 2)  # Create a figure with 2x2 subplots
```

```
fig.set_size_inches(7, 4)
axs[0, 0].plot(x, y1, '-g')  # Plot data on the first subplot
axs[0, 1].plot(x, y2, '--b')  # Plot data on the second subplot
axs[1, 0].plot(x, y3, '-.r')  # Plot data on the third subplot
axs[1, 1].plot(x, y4, ':m')  # Plot data on the fourth subplot

fig.patch.set_alpha(0)
plt.tight_layout()  # Adjust spacing between subplots
plt.show()
```



Visualizing Galaxies data made in chapter 4 Before visiting to the next code try to do the visualization yourself. If you get stuck then refer to the below code.

[3]:
```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('cleaned_data_galaxies.csv')
df.head()
```

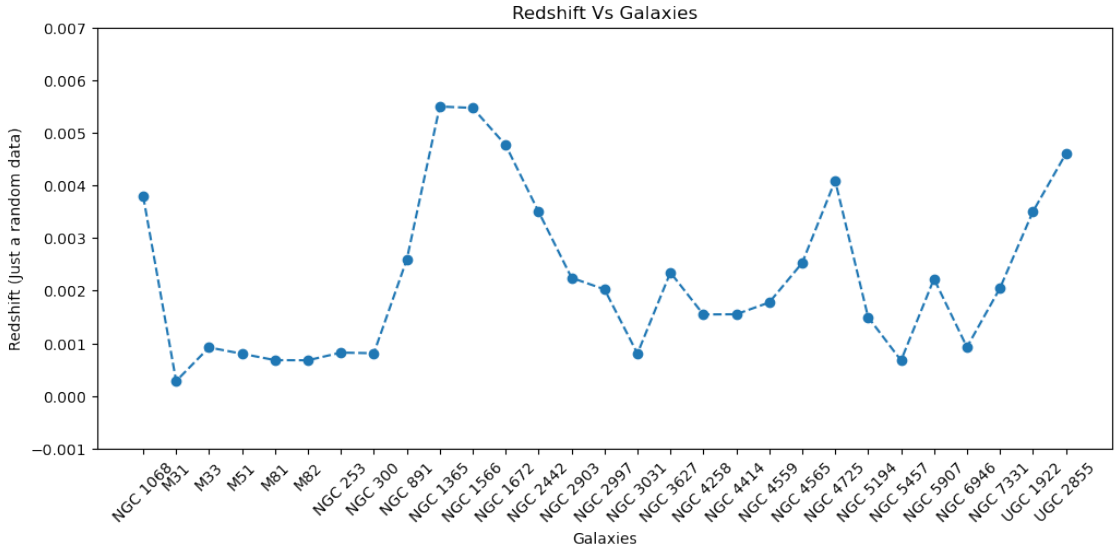
[3]:

	Galaxy	new_Redshift
0	NGC 1068	0.003793
1	M31	0.000282
2	M33	0.000921
3	M51	0.000801
4	M81	0.000680

[4]:
```
length = len(df)
print('We have total',length,'galaxies in our data')
```

```
We have total 29 galaxies in our data
```

[5]:
```
galaxies = df['Galaxy']
z = df['new_Redshift']

plt.figure(figsize=(12,5))
plt.plot(galaxies,z, '--o')
plt.ylabel('Redshift (Just a random data)')
plt.xlabel('Galaxies')

plt.xticks(rotation=45)  # Rotate x-axis labels by 45 degrees to make name␣
 ↪visible
plt.ylim(-0.001, 0.007)  # to extend the limits from -0.001 to 0.007 for the␣
 ↪y-axis to make it clear

plt.title('Redshift Vs Galaxies')
plt.gcf().set_facecolor('none')
plt.show()
```



We have concluded our discussion on Matplotlib, and now, we are poised to delve into the captivating realm of Python for astronomical data analysis—an exciting new phase in our journey!!

Points to remember from the previous chapter:

1. `plt.figure(figsize=(x, y))` is set at the beginning of the code.
2. `plt.gcf().set_facecolor('none')` is positioned just above `plt.show()`.
3. `fig.set_size_inches(x, y)` is utilized in the object-oriented method of subplot creation.
4. `fig.patch.set_alpha(0)` is employed to achieve a transparent background in the object-oriented subplot method.

PART II

Astronomy Data Analysis

1. Astropy

Astropy is a third-party Python module designed to be like a Swiss army knife for the common programming needs of astronomers. Since it is a third-party package, it needs to be installed as mentioned in the first chapter.

Some Features of Astropy:

- Support for typical I/O operations - FITS files, tables, VO, etc.
- Support for basic quantities, astronomical coordinates.
- Statistics and modeling.
- Cosmological calculations.

Astropy Affiliated Packages: Astropy is both the Python module and the community. Within the community, there are many other independently made packages. However, these packages strive to be consistent with the overall organizational theme of Astropy. These packages are called the Astropy Affiliated packages. Some of the packages include:

- ccdproc (for CCD data processing)
- photutils (for performing photometry using a variety of source detection and photometry algorithms)
- specutils (for spectroscopic analyses)
- **astroquery** (for querying commonly used astronomical data archives)
- and many more.

For a full list of all packages, please visit: https://www.astropy.org/affiliated/index.html

THINGS TO BE DONE BY THE END OF THIS SECTION:

1) Playing with astronomical calculations, astronomical coordinates and Gaussian profile
2) Basic FITS exploration
3) SDSS Data Analysis and Agns Classification

The goal of this book is to grill the astroquery portion of an indiviual, so we will focus on astroquery throught the section

1. Playing with astronomical calculations, astronomical coordinates and Gaussian profile

Cosmological Calculations

You no longer need to open a web browser to access a cosmology calculator. You can do all this in Python using astropy's cosmology module. Let's see a quick demonstartion.

[4]:
```
from astropy.cosmology import parameters
print(parameters.available)
```

```
('Planck13', 'Planck15', 'Planck18', 'WMAP1', 'WMAP3', 'WMAP5', 'WMAP7',
'WMAP9')
```

[5]:
```
# Let us choose a model out of one of the above...
from astropy.cosmology import Planck15

print(Planck15.H(1.5)) # what is the Hubble parameter at redshift 1.5?
print(Planck15.Ode(3)) # density parameter for dark energy at redshift z=3
 ↪(in units of critical density)
print(Planck15.critical_density(3))
print(Planck15.Tcmb(1000))
print(Planck15.angular_diameter_distance(2))
print(Planck15.arcsec_per_kpc_comoving(3))
print(Planck15.scale_factor(4))
```

```
159.16501423405055 km / (Mpc s)
0.03373807940562921
1.7653419274193386e-28 g / cm3
2728.2255 K
1770.5128493113361 Mpc
0.03168529715449131 arcsec / kpc
0.2
```

[6]:
```
print("\n".join([i for i in dir(Planck15) if not i[0] == "_"]))
print('Above are attributs we can use within Planck15')
```

```
H
H0
Neff
Ob
Ob0
Ode
Ode0
Odm
Odm0
Ogamma
Ogamma0
Ok
Ok0
```

```
Om
Om0
Onu
Onu0
Otot
Otot0
Tcmb
Tcmb0
Tnu
Tnu0
abs_distance_integrand
absorption_distance
age
angular_diameter_distance
angular_diameter_distance_z1z2
arcsec_per_kpc_comoving
arcsec_per_kpc_proper
clone
comoving_distance
comoving_transverse_distance
comoving_volume
critical_density
critical_density0
de_density_scale
differential_comoving_volume
distmod
efunc
from_format
h
has_massive_nu
hubble_distance
hubble_time
inv_efunc
is_equivalent
is_flat
kpc_comoving_per_arcmin
kpc_proper_per_arcmin
lookback_distance
lookback_time
lookback_time_integrand
luminosity_distance
m_nu
meta
name
nonflat
nu_relative_density
read
scale_factor
```

```
to_format
w
write
Above are attributs we can use within Planck15
```

Example 1: Calculate radial velocity and redshift given the rest wavelenght (λ_0) and observational wavelenght (λ) of the object. To spice things up, you should take into account both relativistic and non-relativistic cases for calculating radial velocity (take $0.1c$ as threshold).

Non-relativistic formula:

$$z = \frac{\lambda - \lambda_0}{\lambda_0} = \frac{\Delta\lambda}{\lambda_0} = \frac{v}{c}$$

Relativistic formula:

$$z = \frac{\Delta\lambda}{\lambda_0} = \sqrt{\frac{1 + v/c}{1 - v/c}} - 1$$

Steps to create an implementation in Python:

- Calculate redshift `z` using observed and rest wavelenghts
- Use the given thrashold to decide which formula to use for calculating `v`. You can use the following Python construction:

```
if condition:
   # relativistic formula
   v = ...
else:
   # non-relativistic formula
   v = ...
```

- print the values of `v` and `z` (nicely formated)

https://docs.astropy.org/en/stable/units/equivalencies.html#spectral-doppler-equivalencies

[8]:
```
import astropy.constants as const
import astropy.units as u
```

[11]:
```
def redshift(lam_obs, lam_rest):
  """
  A function to calculate velocity and redshift
  Inputs:
  ------
  lam_obs, lam_rest: observed and rest wavelengths
  Returns :
  -------
  v: velocity, z: redshift
  """
```

```
  freq_obs = (lam_obs * u.nm).to(u.Hz, equivalencies=u.spectral())
  freq_rest = (lam_rest * u.nm).to(u.Hz, equivalencies=u.spectral())

  z = (freq_rest - freq_obs)/freq_rest

  if z < 0.1:
    v = z * const.c
    return z, v

  else:
    freq_to_vel = u.doppler_relativistic(freq_rest)
    v = (freq_obs).to(u.km / u.s, equivalencies=freq_to_vel)
    return z, v


z, v = redshift(1500, 1100)

print(f"The object is moving with a velocity {v} and has a redshift {z}")
```

```
The object is moving with a velocity 90111.02783815032 km / s and has a redshift
0.2666666666666668
```

Example 2: Write a function that calculates a distance to a star, given its apparent magnitude and absolute magnitude, in units of light-years.

[15]:
```
import astropy.units as u

def distance(m_app, m_abs):
  """
  A function to calculate distance to a star given apparent and absolute␣
 ↪magnitudes.
  We do not correct for extinction.
  Inputs:
  ------
  m_app: apparent magnitude, m_abs: absolute magnitude
  Returns :
  -------
  d_ly: distance in light years
  """

  dm = m_app - m_abs
  d_pc = 10 * 10**(dm/5)

  d_ly = (d_pc * u.parsec).to(u.lyr)

  return d_ly

distance(21, 16)
```

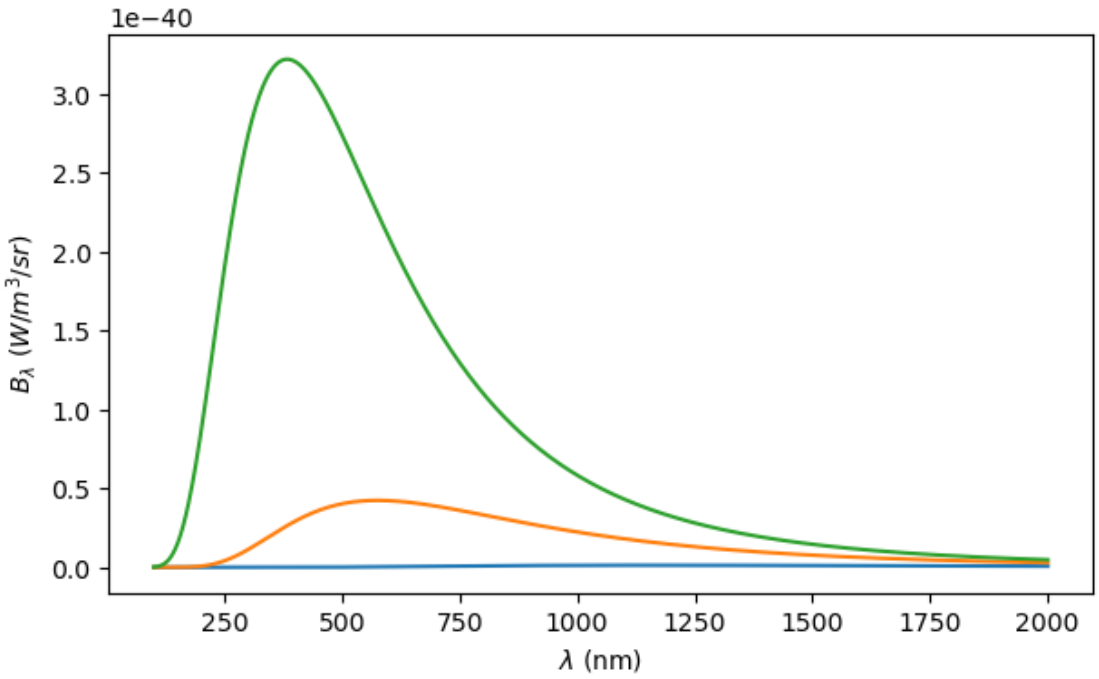
[15]:
326.15638 lyr

Example 3: Implement a function that calculates spectral radiance using the Planck's law. The function should have two input parameters: temperature and wavelength. Plot the spectral radiance as a function of wavelength for several values of temperature.

https://physics.info/planck/

[16]:
```
import matplotlib.pyplot as plt
import numpy as np
import astropy.constants as cont
import astropy.units as u
```

[33]:
```
plt.figure(figsize=(7,4))

def planck(lam, T):
    """
    A function to calculate the spectral radiance using the Planck's law
    Inputs:
    ------
    lam: wavelength and T: temperature
    Returns :
    -------
    B_lam: spectral radiance
    """

    # add units to inputs


    B_lam = 2 * const.h * const.c / ((lam * u.nm)**5 * np.e**((const.h * const.
 ↪c)/ ((lam * u.nm) * const.k_B * (T * u.Kelvin))))

    return B_lam

lam = np.linspace(100, 2000, 1000)

temp = [2500, 5000, 7500]

for T in temp:
    plt.plot(lam, planck(lam, T), label=str(T)+"K")

plt.xlabel(r"$\lambda$ (nm)")
plt.ylabel(r"$B_{\lambda}$ $(W/m^3/sr)$")
plt.show()
```



Example 4: ISO Date to Julian Date

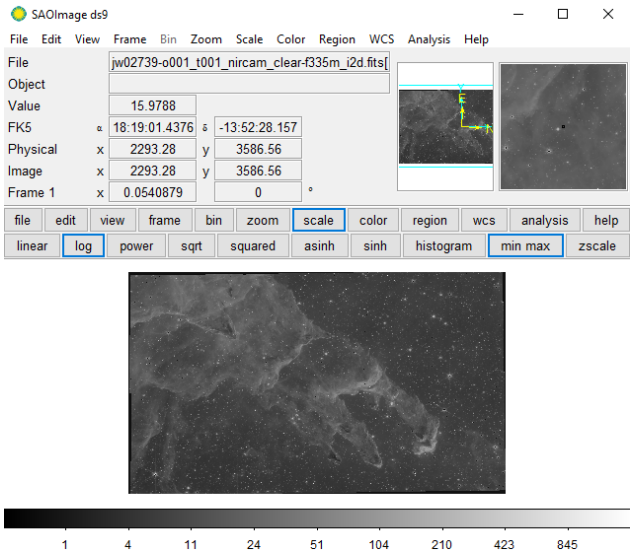
[22]:
```
from astropy.time import Time

# Define the date in ISO format (YYYY-MM-DD)
date = input("Enter the date in ISO format (YYYY-MM-DD): \n ")

# Create an Astropy Time object
time = Time(date, format='iso', scale='utc')

# Convert the time to Modified Julian Date (MJD)
mjd = time.mjd

print(f"The Modified Julian Date (MJD) for {date} is: \n {mjd:.5f}")
```

```
Enter the date in ISO format (YYYY-MM-DD):
  2023-08-11
```

```
The Modified Julian Date (MJD) for 2023-08-11 is:
 60167.00000
```

World Coordinate system in Astropy

If you open an image in ds9 or any other image viewer, you will notice that as the mouse pointer moves across various regions in the image, a part of the window will reflect the current coordinates in some coordinate system. This is possible because the image header contains a whole bunch of keywords which can be used to compute the coordinates as a function of the pixel coordinates.



What one needs is the coordinate for one pixel and then variation in coordinates for every pixel of movement. But if the image is very large, the curvature of the sky can become important. Then additional information to account for second order effects will need to be recorded

Let's see how a good fraction of all this is handled using Astropy.

Simple steps:

```
Get hold of the header.
Pass it to a WCS object constructor.
And use this object for coordinate conversions.
```

[16]:
```
from astropy.io import fits
import astropy.wcs as wcs

hdulist = fits.open('example2.fits')
w = wcs.WCS(hdulist[0].header)
print(w)
```

```
WCS Keywords

Number of WCS axes: 2
CTYPE : 'RA---TAN'  'DEC--TAN'
CRVAL : 205.4864335  47.26686298333333
CRPIX : -3929.556588388735  377.0604456526553
PC1_1 PC1_2  : 0.02528041441746252  2.266833305288208e-05
PC2_1 PC2_2  : -1.983416356621748e-05  0.02528846844193638
```

```
CDELT : -0.01868178756738699  0.01868178756738699
NAXIS : 777  777
```

```
WARNING: FITSFixedWarning: 'datfix' made the change 'Set MJD-OBS to 35571.000000
from DATE-OBS.
Changed DATE-OBS from '08/04/56          ' to '1956-04-08''. [astropy.wcs.wcs]
```

[17]:
```
# Convert pixels to coordinates.
w.wcs_pix2world(100, 100, 1)
```

[17]:
```
[array(202.69037279), array(47.10039636)]
```

[18]:
```
w.wcs_pix2world([100, 105], [100, 105], 1)
```

[18]:
```
[array([202.69037279, 202.68677899]), array([47.10039636, 47.10266995])]
```

[19]:
```
w.wcs_world2pix(202.69, 47.1, 1)
```

[19]:
```
[array(100.56778084), array(99.1810579)]
```

[20]:
```
w.calc_footprint()
```

[20]:
```
array([[202.76146626,  47.05535561],
       [202.74206475,  47.42164629],
       [202.2009436 ,  47.40718098],
       [202.22406523,  47.04102483]])
```

Some convenience functions

There are some utilities provided for your convenience which make coordinate to pixel and vice versa calculations much simple.

[21]:
```
output = wcs.utils.pixel_to_skycoord(100, 100, w)
output
```

[21]:
```
<SkyCoord (FK5: equinox=2000.0): (ra, dec) in deg
    (202.68965405, 47.10085109)>
```

Remember: This is capable of ignoring higher dimensions in WCS in radio data, for example.

The output here is a special object known as a sky coordinate object. This can be created by saying,

```
from astropy.coordinates import SkyCoord
c = SkyCoord(10, 20, unit='deg')
```

[22]:
```
import astropy

new = output.transform_to(astropy.coordinates.Galactic)
new
```

[22]:
```
<SkyCoord (Galactic): (l, b) in deg
    (104.37183205, 68.58518637)>
```

[23]:
```
from astropy.coordinates import SkyCoord as sc

distance = output.separation(sc(202, 47, unit='deg'))
print(distance)
```

```
0d28m50.15243683s
```

[24]:
```
output.ra, output.dec
```

[24]:
```
(<Longitude 202.68965405 deg>, <Latitude 47.10085109 deg>)
```

Data Modeling with Astropy

"modeling" is a library in Python designed to give you

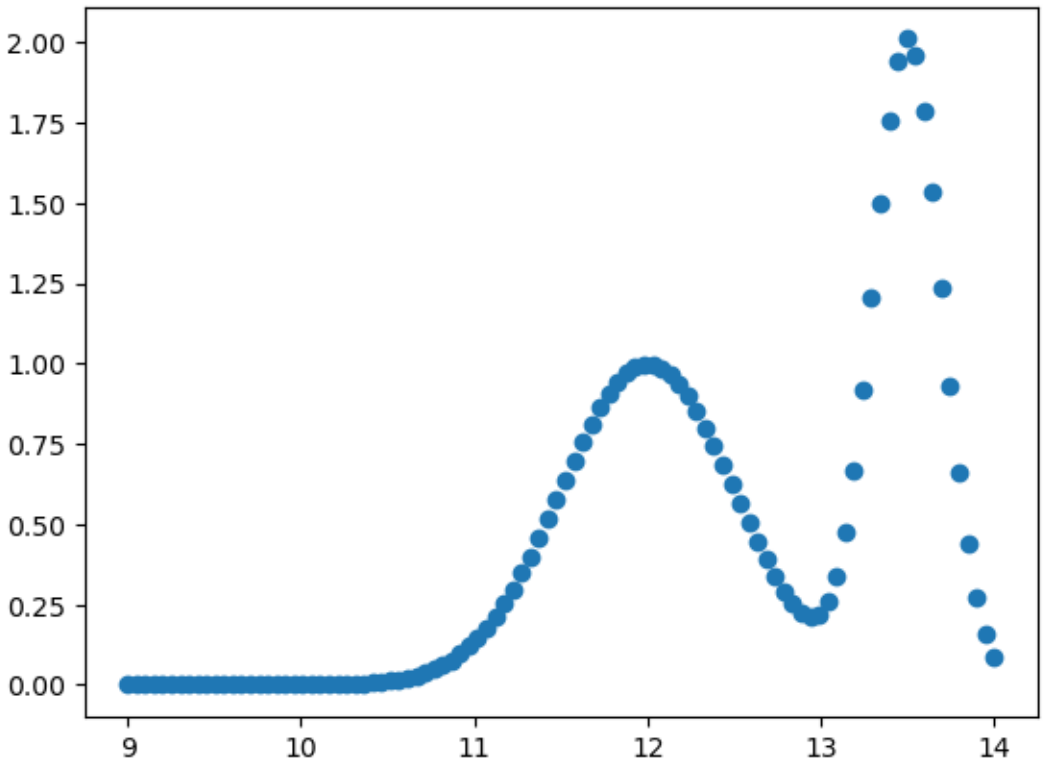
- Access to commonly used models.
- As well as fit them to various data.

Let's see how to use this module.

First off all, we will use the models prebuilt into this module to generate some data.

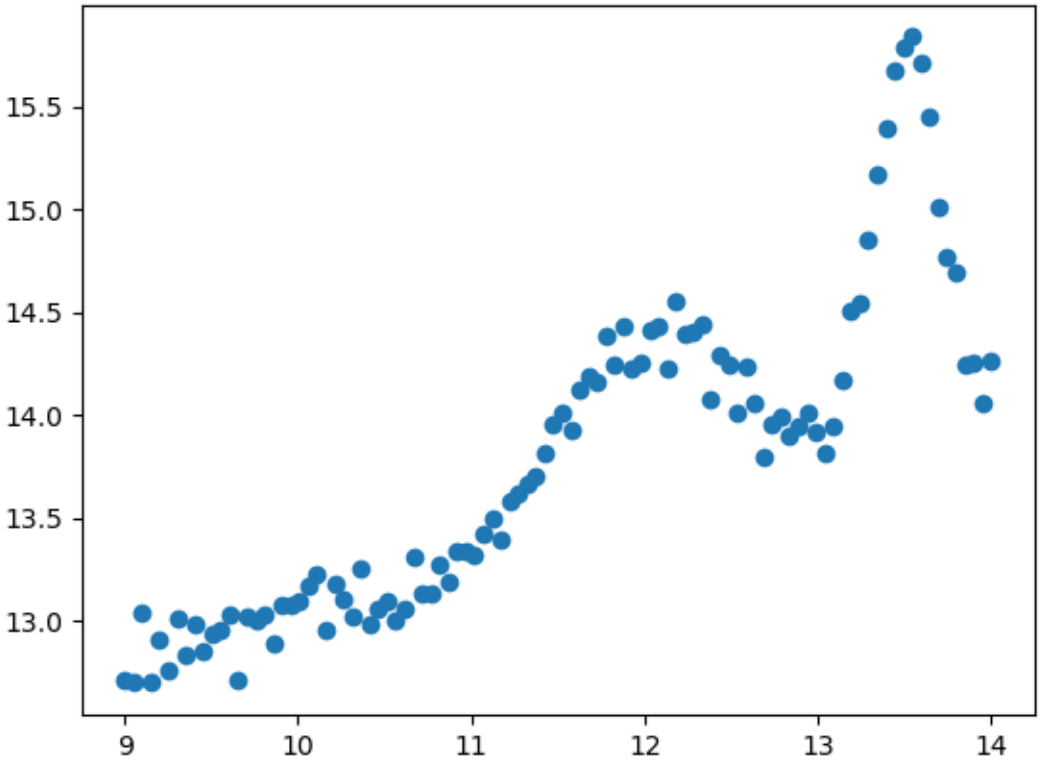
[27]:
```
from astropy.modeling import models
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

x = np.linspace(9, 14, 100)
gauss_example1 = models.Gaussian1D(amplitude=1.0, mean=12, stddev=0.5)
gauss_example2 = models.Gaussian1D(amplitude=2.0, mean=13.5, stddev=0.2)
gauss_total = gauss_example1 + gauss_example2
y = gauss_total(x)

plt.scatter(x,y)
```

[27]:
```
<matplotlib.collections.PathCollection at 0x237bec7e3d0>
```



[29]:
```
import numpy.random as npr

y_noise = npr.normal(0, 0.1, len(x))
y_obs = 12 + 0.01*x**2 + y + y_noise
plt.scatter(x, y_obs)
```

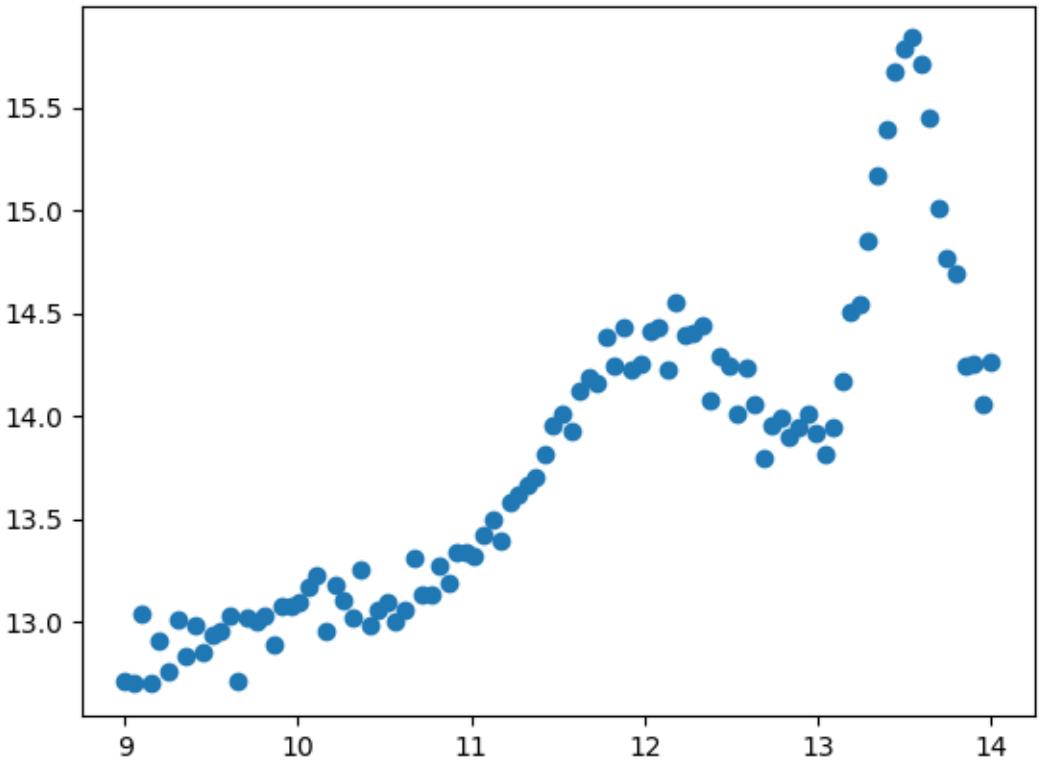
[29]:
```
<matplotlib.collections.PathCollection at 0x237bed9d8d0>
```



We saw how trivial it is to use a model and actually evaluate it over a range. But a more useful thing we need to do with models is to fit some data. Now, pretend that x and y_obs are two arrays that contain our observed data. We can start by plotting the same.

[31]:
```
plt.scatter(x, y_obs)
```

[31]:
```
<matplotlib.collections.PathCollection at 0x237bee0c290>
```



Once we have a plot, our next step is to choose a model. What model shall fit the data well? Let us assume that these are spectral features with some known wavelengths. So, we may assume that each feature is a Gaussian. To minimize number if independent parameters, we may also consider that the difference in the wavelengths is a constant. Such a constraint obviously comes from our prior knowledge of the physical system that generated the spectrum.

Next, these emission features clearly are not on their own - they are stop a continuum. Assuming that the continuum is not varying at a fast rate wrt wavelengths (X-axis), we can further assume that a quadratic polynomial suffices in accounting for this.

So, our model is a second order polynomial plus two Gaussians, with different means but separated by a known amount.

[32]:
```
# So, let us construct our model.
model = models.Gaussian1D(amplitude=1.0, mean=12.1, stddev=0.5) +\
        models.Gaussian1D(amplitude=1.0, mean=13.6, stddev=0.4) +\
        models.Polynomial1D(degree=2)
```

[33]:
```
print(model.param_names)
```

```
('amplitude_0', 'mean_0', 'stddev_0', 'amplitude_1', 'mean_1', 'stddev_1',
'c0_2', 'c1_2', 'c2_2')
```

[34]:
```
# Our model is not complete. We must supply our constraint.
def constraint_mean(model):
    mean_0 = model.mean_1 - 1.5
    return mean_0

model.mean_0.tied = constraint_mean
```

The model is ready. We have the data. What we finally need is a fitting algorithm. Let us choose the Marquardt Levenberg method.

[36]:
```
from astropy.modeling import fitting
fitter = fitting.LevMarLSQFitter()

model_fit = fitter(model, x, y_obs)
```

[37]:
```
model_fit.parameters
```

[37]:
```
array([ 9.19306149e-01,  1.20077964e+01,  4.90426705e-01,  1.91281771e+00,
        1.35077964e+01,  1.88374457e-01,  1.20955187e+01, -2.42277967e-02,
        1.16508691e-02])
```

[38]:
```
model_fit.param_names
```

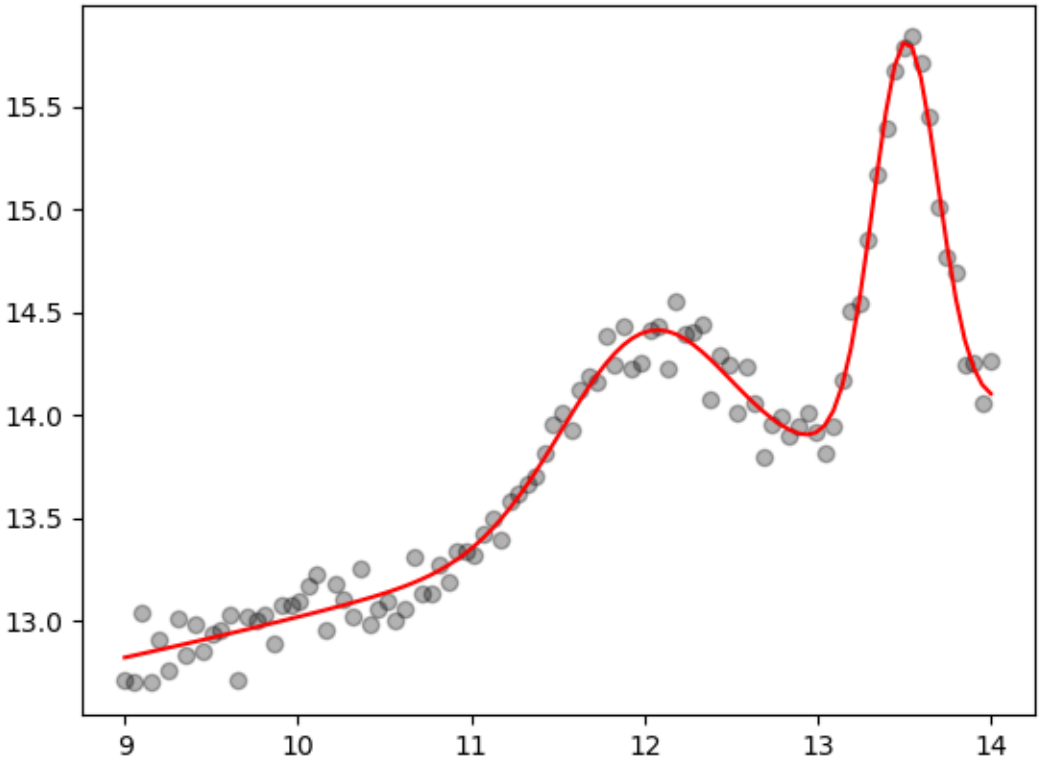
[38]:
```
('amplitude_0',
 'mean_0',
 'stddev_0',
 'amplitude_1',
 'mean_1',
 'stddev_1',
 'c0_2',
 'c1_2',
 'c2_2')
```

[39]:
```
dict(zip(model_fit.param_names, model_fit.parameters))
```

[39]:
```
{'amplitude_0': 0.9193061494110074,
 'mean_0': 12.007796408435455,
 'stddev_0': 0.49042670519457093,
 'amplitude_1': 1.9128177135225632,
 'mean_1': 13.507796408435455,
 'stddev_1': 0.1883744574702892,
 'c0_2': 12.095518674457615,
 'c1_2': -0.0242277966822873,
 'c2_2': 0.0116508690750115}
```

[40]:
```
plt.scatter(x, y_obs, color='black', alpha=0.3)
plt.plot(x, model_fit(x), color='red')
```

[40]: [<matplotlib.lines.Line2D at 0x237bec410d0>]



2. Basic FITS exploration

FITS Files in Astropy

What's FITS!?

FITS stands for "textbf{Flexible Image Transport System}". It is a file format designed by astronomers. And it is indeed flexible - it can store images (as 2-d arrays) and it can store tables (ASCII and binary). And a given file can have both - this allows one to attach tabular data directly to images. eg. a catalog of photometry can be attached to the image from which it was extracted. You can also store multi-dimensional data and more.

A FITS file is made of one or more header/data units (HDUs).

```
Each HDU is made of a header unit which contains metadata in the form of a key-value pair. This
An optional component within a HDU is the data unit which contains the data unit.
```

Astropy has a sub-module for handling FITS files. Let's see how to load a file.

[165]:
```
# First import the sub-module
from astropy.io import fits

# To load a FITS file
hdulist = fits.open('example.fits')

# This is a special type of list.
# Supports more or less all list operations.
# And some more (like writeto)
print(type(hdulist))
```

```
<class 'astropy.io.fits.hdu.hdulist.HDUList'>
```

[166]:
```
# HDUList contains HDU objects.
print(type(hdulist[0]))
```

```
<class 'astropy.io.fits.hdu.image.PrimaryHDU'>
```

When dealing with a new file, the first obvious step is to examine the file and understand how data is organized in it.

[167]:
```
print(hdulist.info())
```

```
Filename: example.fits
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU      71   (512, 512)   int16
None
```

[168]:
```
header = hdulist[0].header
print(header.cards)
```

```
('SIMPLE', True, 'Fits standard')
('BITPIX', 16, 'Bits per pixel')
('NAXIS', 2, 'Number of axes')
```

```
('NAXIS1', 512, 'Axis length')
('NAXIS2', 512, 'Axis length')
('EXTEND', False, 'File may contain extensions')
('ORIGIN', 'NOAO-IRAF FITS Image Kernel July 2003', 'FITS file originator')
('DATE', '11-08-2016', 'Date FITS file was generated')
('IRAF-TLM', '2017-02-17T04:36:31', 'Time of last modification')
('OBJECT', 'm51  B  600s', 'Name of the object observed')
('IRAF-MAX', 19936.0, 'DATA MAX')
('IRAF-MIN', -1.0, 'DATA MIN')
('CCDPICNO', 53, 'ORIGINAL CCD PICTURE NUMBER')
('ITIME', 600, 'REQUESTED INTEGRATION TIME (SECS)')
('TTIME', 600, 'TOTAL ELAPSED TIME (SECS)')
('OTIME', 600, 'ACTUAL INTEGRATION TIME (SECS)')
('DATA-TYP', 'OBJECT (0)', 'OBJECT,DARK,BIAS,ETC.')
('DATE-OBS', '05/04/87', 'DATE DD/MM/YY')
('RA', '13:29:24.00', 'RIGHT ASCENSION')
('DEC', '47:15:34.00', 'DECLINATION')
('EPOCH', 0.0, 'EPOCH OF RA AND DEC')
('ZD', '22:14:00.00', 'ZENITH DISTANCE')
('UT', ' 9:27:27.00', 'UNIVERSAL TIME')
('ST', '14:53:42.00', 'SIDEREAL TIME')
('CAM-ID', 1, 'CAMERA HEAD ID')
('CAM-TEMP', -106.22, 'CAMERA TEMPERATURE, DEG C')
('DEW-TEMP', -180.95, 'DEWAR TEMPRATURE, DEG C')
('F1POS', 2, 'FILTER BOLT I POSITION')
('F2POS', 0, 'FILTER BOLT II POSITION')
('TVFILT', 0, 'TV FILTER')
('CMP-LAMP', 0, 'COMPARISON LAMP')
('TILT-POS', 0, 'TILT POSITION')
('BIAS-PIX', 0, '')
('BI-FLAG', 0, 'BIAS SUBTRACT FLAG')
('BP-FLAG', 0, 'BAD PIXEL FLAG')
('CR-FLAG', 0, 'BAD PIXEL FLAG')
('DK-FLAG', 0, 'DARK SUBTRACT FLAG')
('FR-FLAG', 0, 'FRINGE FLAG')
('FR-SCALE', 0.0, 'FRINGE SCALING PARAMETER')
('TRIM', 'Apr 22 14:11 Trim image section is [3:510,3:510]', '')
('BT-FLAG', 'Apr 22 14:11 Overscan correction strip is [515:544,3:510]', '')
('FF-FLAG', 'Apr 22 14:11 Flat field image is Flat1.imh with scale=183.9447',
'')
('CCDPROC', 'Apr 22 14:11 CCD processing done', '')
('AIRMASS', 1.08015632629395, 'AIRMASS')
('OBSERVER', 'Jashanpreet', '')
('CREDIT', 'NASA', '')
('MESSAGE', 'Today is 11th August', 'I LOVE ASTRONOMY.')
('HISTORY', "'KPNO-IRAF'", '')
('HISTORY', "'24-04-87'", '')
('HISTORY', "'KPNO-IRAF'           /", '')
```

```
('HISTORY', "'08-04-92'            /", '')
('', '', '')
('', '', '')
('', '', '')
('', '', '')
('', '', '')
('', '', '')
('', '', '')
('', '', '')
('', '', '')
('', '', '')
('', '', '')
('', '', '')
('', '', '')
('', '', '')
('', '', '')
('', '', '')
('', '', '')
('', '', '')
('', '', '')
('', '', '')
('', '', '')
```

Header objects behave almost like dictionaries. So, you can query values and change them using a dictionary like syntax.

[169]:
```
print(header['NAXIS1'])
```

```
512
```

[170]:
```
print(header['NAXIS2'])
```

```
512
```

[171]:
```
#edit the fits header
header['INSTRMNT'] = 'MyTelescope'
header['Bday']='11/08/2006'

#A secure way to edit fits header correctly
fits.setval('example.fits', 'OBSERVER', value='Jashanpreet')
fits.setval('example.fits', 'Date', value='11-08-2016')
fits.setval('example.fits', 'Credit', value='NASA')

fits.setval('example.fits', 'Message', value='Today is 11th August',comment="I␣
 ↪LOVE ASTRONOMY.")
```

[172]:
```
header = hdulist[0].header
print(header.cards)
```

```
('SIMPLE', True, 'Fits standard')
('BITPIX', 16, 'Bits per pixel')
```

```
('NAXIS', 2, 'Number of axes')
('NAXIS1', 512, 'Axis length')
('NAXIS2', 512, 'Axis length')
('EXTEND', False, 'File may contain extensions')
('ORIGIN', 'NOAO-IRAF FITS Image Kernel July 2003', 'FITS file originator')
('DATE', '11-08-2016', 'Date FITS file was generated')
('IRAF-TLM', '2017-02-17T04:36:31', 'Time of last modification')
('OBJECT', 'm51  B  600s', 'Name of the object observed')
('IRAF-MAX', 19936.0, 'DATA MAX')
('IRAF-MIN', -1.0, 'DATA MIN')
('CCDPICNO', 53, 'ORIGINAL CCD PICTURE NUMBER')
('ITIME', 600, 'REQUESTED INTEGRATION TIME (SECS)')
('TTIME', 600, 'TOTAL ELAPSED TIME (SECS)')
('OTIME', 600, 'ACTUAL INTEGRATION TIME (SECS)')
('DATA-TYP', 'OBJECT (0)', 'OBJECT,DARK,BIAS,ETC.')
('DATE-OBS', '05/04/87', 'DATE DD/MM/YY')
('RA', '13:29:24.00', 'RIGHT ASCENSION')
('DEC', '47:15:34.00', 'DECLINATION')
('EPOCH', 0.0, 'EPOCH OF RA AND DEC')
('ZD', '22:14:00.00', 'ZENITH DISTANCE')
('UT', ' 9:27:27.00', 'UNIVERSAL TIME')
('ST', '14:53:42.00', 'SIDEREAL TIME')
('CAM-ID', 1, 'CAMERA HEAD ID')
('CAM-TEMP', -106.22, 'CAMERA TEMPERATURE, DEG C')
('DEW-TEMP', -180.95, 'DEWAR TEMPRATURE, DEG C')
('F1POS', 2, 'FILTER BOLT I POSITION')
('F2POS', 0, 'FILTER BOLT II POSITION')
('TVFILT', 0, 'TV FILTER')
('CMP-LAMP', 0, 'COMPARISON LAMP')
('TILT-POS', 0, 'TILT POSITION')
('BIAS-PIX', 0, '')
('BI-FLAG', 0, 'BIAS SUBTRACT FLAG')
('BP-FLAG', 0, 'BAD PIXEL FLAG')
('CR-FLAG', 0, 'BAD PIXEL FLAG')
('DK-FLAG', 0, 'DARK SUBTRACT FLAG')
('FR-FLAG', 0, 'FRINGE FLAG')
('FR-SCALE', 0.0, 'FRINGE SCALING PARAMETER')
('TRIM', 'Apr 22 14:11 Trim image section is [3:510,3:510]', '')
('BT-FLAG', 'Apr 22 14:11 Overscan correction strip is [515:544,3:510]', '')
('FF-FLAG', 'Apr 22 14:11 Flat field image is Flat1.imh with scale=183.9447',
'')
('CCDPROC', 'Apr 22 14:11 CCD processing done', '')
('AIRMASS', 1.08015632629395, 'AIRMASS')
('OBSERVER', 'Jashanpreet', '')
('CREDIT', 'NASA', '')
('MESSAGE', 'Today is 11th August', 'I LOVE ASTRONOMY.')
('INSTRMNT', 'MyTelescope', '')
('BDAY', '11/08/2006', '')
```

```
('HISTORY', "'KPNO-IRAF'", '')
('HISTORY', "'24-04-87'", '')
('HISTORY', "'KPNO-IRAF'           /", '')
('HISTORY', "'08-04-92'            /", '')
('', '', '')
('', '', '')
('', '', '')
('', '', '')
('', '', '')
('', '', '')
('', '', '')
('', '', '')
('', '', '')
('', '', '')
('', '', '')
('', '', '')
('', '', '')
('', '', '')
('', '', '')
('', '', '')
('', '', '')
('', '', '')
```

You can see that keywords are adding successfully to the fits header

The data can be accessed using the data attribute of the HDU.

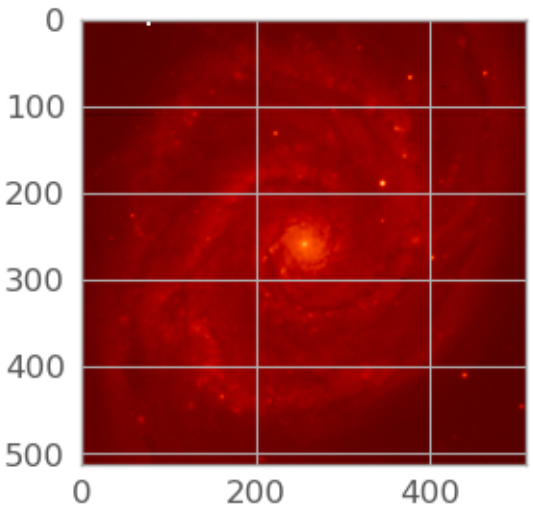
[76]:
```
data = hdulist[0].data
print(type(data))
```

```
<class 'numpy.ndarray'>
```

[142]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
plt.figure(figsize=(3,3))
plt.imshow(np.log(data))
```

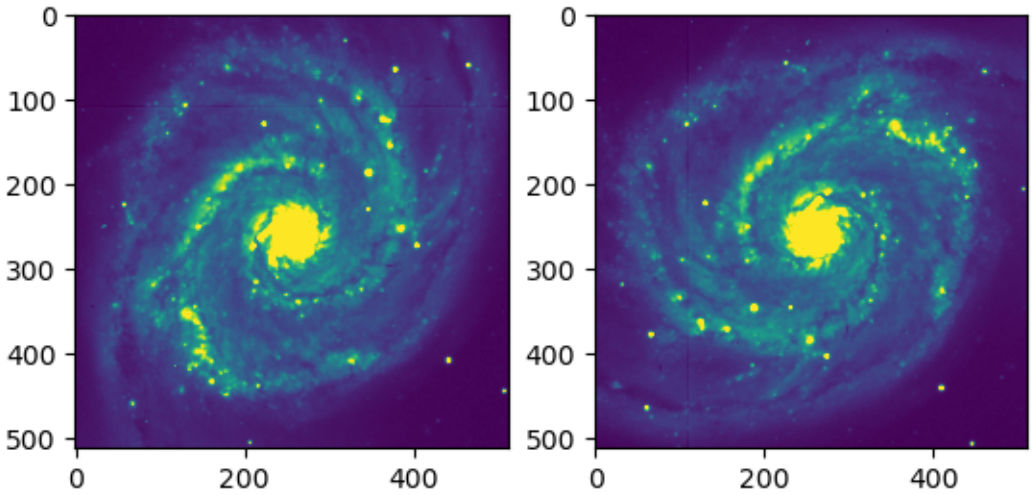
```
C:\Users\Admin\AppData\Local\Temp\ipykernel_12024\1860379595.py:5:
RuntimeWarning: invalid value encountered in log
  plt.imshow(np.log(data))
```

[142]:
```
<matplotlib.image.AxesImage at 0x237ca9cc6d0>
```



[80]:
```
# Images in ZScale

from astropy.visualization import ZScaleInterval
z = ZScaleInterval()
z1, z2 = z.get_limits(data)
fig, ax = plt.subplots(nrows=1, ncols=2)
ax[0].imshow(data, vmin=z1, vmax=z2)
ax[1].imshow(data.T, vmin=z1, vmax=z2)
```

[80]: <matplotlib.image.AxesImage at 0x237c1458c50>



Making a FITS file

[82]:
```
import numpy.random as npr
data_ = npr.normal(0,1,262144).reshape(512, 512)
#data_ is given so that the name should not match with earlier code and create␣
 ↪error in future

hdulist = fits.HDUList()
hdu1 = fits.PrimaryHDU()
hdu1.data = data_
hdulist.append(hdu1)

hdulist.writeto('Myfirstfits.fits',overwrite=True)
```

[83]:
```
col1 = fits.Column(name='col1', format='E', array=nr.normal(0, 1, 10))
col2 = fits.Column(name='col2', format='E', array=nr.normal(3, 1, 10))
cols = fits.ColDefs([col1, col2])
hdu2 = fits.BinTableHDU.from_columns(cols)
hdulist.append(hdu2)
hdulist.writeto('Mysecondfits.fits',overwrite=True) #You can remove␣
 ↪overwrite=True if there is not fits file with same name already in the␣
 ↪working folder
```

Convenience Functions

Astropy's io.fits library comes with a whole bunch of convenience functions. This makes common operations with FITS file much easier.

```
header = fits.getheader('somefile.fits')
naxis1 = fits.getval('somefile.fits', 'naxis1', 1)
data = fits.getdata('some.fits', 2)

fits.writeto('some.fits', data, header)
fits.append('some.fits', data, header)
fits.update('some.fits, data, header, 1)
```

TASK 2: Study in details the following data: http://www.astropy.org/astropy-data/tutorials/FITS-images/HorseHead.fits Study the fits header, edit it, plot the histogram of the data, use cmap to plot the image.

TASK 2 SOLUTION

[2]:
```
from astropy.visualization import astropy_mpl_style
import matplotlib.pyplot as plt
import numpy as np
# Important
from astropy.io import fits
from astropy.utils.data import download_file
```

[3]:
```
plt.style.use(astropy_mpl_style)
image_file = download_file('http://www.astropy.org/astropy-data/tutorials/
↪FITS-images/HorseHead.fits', cache=True)
```

Let's open the FITS file to find out what it contains.

[4]:
```
hdu_list = fits.open(image_file)
hdu_list.info()
```

```
Filename: C:\Users\Admin\.astropy\cache\download\url\217b4fe80e6f349ef703ceed7e0
be888\contents
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU     163   (891, 893)   int16
  1  er.mask       1 TableHDU        25   1600R x 4C   [F6.2, F6.2, F6.2, F6.2]
```

Generally, the image information is located in the PRIMARY block. The blocks are numbered and can be accessed by indexing hdu_list.

[8]:
```
hdu_list[0].header
```

[8]:
```
SIMPLE  =                    T /FITS: Compliance
BITPIX  =                   16 /FITS: I*2 Data
NAXIS   =                    2 /FITS: 2-D Image Data
NAXIS1  =                  891 /FITS: X Dimension
NAXIS2  =                  893 /FITS: Y Dimension
EXTEND  =                    T /FITS: File can contain extensions
DATE    = '11-08-2016'         / FITS: Creation Date
ORIGIN  = 'STScI/MAST'         /GSSS: STScI Digitized Sky Survey
SURVEY  = 'SERC-ER '           /GSSS: Sky Survey
REGION  = 'ER768   '           /GSSS: Region Name
PLATEID = 'A0JP    '           /GSSS: Plate ID
SCANNUM = '01      '           /GSSS: Scan Number
DSCNDNUM= '00      '           /GSSS: Descendant Number
TELESCID=                    4 /GSSS: Telescope ID
BANDPASS=                   36 /GSSS: Bandpass Code
COPYRGHT= 'AAO/ROE '           /GSSS: Copyright Holder
SITELAT =              -31.277 /Observatory: Latitude
SITELONG=              210.934 /Observatory: Longitude
TELESCOP= 'UK Schmidt - Doubl' /Observatory: Telescope
INSTRUME= 'Photographic Plate' /Detector: Photographic Plate
```

```
EMULSION= 'IIIaF   '           /Detector: Emulsion
FILTER  = 'OG590   '           /Detector: Filter
PLTSCALE=               67.20 /Detector: Plate Scale arcsec per mm
PLTSIZEX=             355.000 /Detector: Plate X Dimension mm
PLTSIZEY=             355.000 /Detector: Plate Y Dimension mm
PLATERA =        85.5994550000 /Observation: Field centre RA degrees
PLATEDEC=       -4.94660910000 /Observation: Field centre Dec degrees
PLTLABEL= 'OR14052 '           /Observation: Plate Label
DATE-OBS= '1990-12-22T13:49:00' /Observation: Date/Time
EXPOSURE=                65.0 /Observation: Exposure Minutes
PLTGRADE= 'AD2     '           /Observation: Plate Grade
OBSHA   =            0.158333 /Observation: Hour Angle
OBSZD   =             26.3715 /Observation: Zenith Distance
AIRMASS =             1.11587 /Observation: Airmass
REFBETA =        66.3196420000 /Observation: Refraction Coeff
REFBETAP=     -0.0820000000000 /Observation: Refraction Coeff
REFK1   =        6423.52290000 /Observation: Refraction Coeff
REFK2   =       -102122.550000 /Observation: Refraction Coeff
CNPIX1  =               12237 /Scan: X Corner
CNPIX2  =               19965 /Scan: Y Corner
XPIXELS =               23040 /Scan: X Dimension
YPIXELS =               23040 /Scan: Y Dimension
XPIXELSZ=             15.0295 /Scan: Pixel Size microns
YPIXELSZ=             15.0000 /Scan: Pixel Size microns
PPO1    =       -3069417.00000 /Scan: Orientation Coeff
PPO2    =       0.000000000000 /Scan: Orientation Coeff
PPO3    =        177500.000000 /Scan: Orientation Coeff
PPO4    =       0.000000000000 /Scan: Orientation Coeff
PPO5    =        3069417.00000 /Scan: Orientation Coeff
PPO6    =        177500.000000 /Scan: Orientation Coeff
PLTRAH  =                   5 /Astrometry: Plate Centre H
PLTRAM  =                  42 /Astrometry: Plate Centre M
PLTRAS  =               23.86 /Astrometry: Plate Centre S
PLTDECSN= '-       '           /Astrometry: Plate Centre +/-
PLTDECD =                   4 /Astrometry: Plate Centre D
PLTDECM =                  56 /Astrometry: Plate Centre M
PLTDECS =                47.9 /Astrometry: Plate Centre S
EQUINOX =              2000.0 /Astrometry: Equinox
AMDX1   =        67.1550859799 /Astrometry: GSC1 Coeff
AMDX2   =      0.0431478884485 /Astrometry: GSC1 Coeff
AMDX3   =       -292.435619180 /Astrometry: GSC1 Coeff
AMDX4   =  -2.68934864702E-005 /Astrometry: GSC1 Coeff
AMDX5   =   1.99133423290E-005 /Astrometry: GSC1 Coeff
AMDX6   =  -2.37011931379E-006 /Astrometry: GSC1 Coeff
AMDX7   =       0.000000000000 /Astrometry: GSC1 Coeff
AMDX8   =   2.21426387429E-006 /Astrometry: GSC1 Coeff
AMDX9   =  -8.12841581455E-008 /Astrometry: GSC1 Coeff
```

```
AMDX10  =   2.48169090021E-006 /Astrometry: GSC1 Coeff
AMDX11  =   2.77618933926E-008 /Astrometry: GSC1 Coeff
AMDX12  =       0.000000000000 /Astrometry: GSC1 Coeff
AMDX13  =       0.000000000000 /Astrometry: GSC1 Coeff
AMDX14  =       0.000000000000 /Astrometry: GSC1 Coeff
AMDX15  =       0.000000000000 /Astrometry: GSC1 Coeff
AMDX16  =       0.000000000000 /Astrometry: GSC1 Coeff
AMDX17  =       0.000000000000 /Astrometry: GSC1 Coeff
AMDX18  =       0.000000000000 /Astrometry: GSC1 Coeff
AMDX19  =       0.000000000000 /Astrometry: GSC1 Coeff
AMDX20  =       0.000000000000 /Astrometry: GSC1 Coeff
AMDY1   =        67.1593591466 /Astrometry: GSC1 Coeff
AMDY2   =     -0.0471363749174 /Astrometry: GSC1 Coeff
AMDY3   =        316.004963520 /Astrometry: GSC1 Coeff
AMDY4   =   2.86798151430E-005 /Astrometry: GSC1 Coeff
AMDY5   =  -2.00968236347E-005 /Astrometry: GSC1 Coeff
AMDY6   =   2.27840393227E-005 /Astrometry: GSC1 Coeff
AMDY7   =       0.000000000000 /Astrometry: GSC1 Coeff
AMDY8   =   2.23885090381E-006 /Astrometry: GSC1 Coeff
AMDY9   =  -2.28360163464E-008 /Astrometry: GSC1 Coeff
AMDY10  =   2.44828851495E-006 /Astrometry: GSC1 Coeff
AMDY11  =  -5.76717487998E-008 /Astrometry: GSC1 Coeff
AMDY12  =       0.000000000000 /Astrometry: GSC1 Coeff
AMDY13  =       0.000000000000 /Astrometry: GSC1 Coeff
AMDY14  =       0.000000000000 /Astrometry: GSC1 Coeff
AMDY15  =       0.000000000000 /Astrometry: GSC1 Coeff
AMDY16  =       0.000000000000 /Astrometry: GSC1 Coeff
AMDY17  =       0.000000000000 /Astrometry: GSC1 Coeff
AMDY18  =       0.000000000000 /Astrometry: GSC1 Coeff
AMDY19  =       0.000000000000 /Astrometry: GSC1 Coeff
AMDY20  =       0.000000000000 /Astrometry: GSC1 Coeff
AMDREX1 =        67.1532034737 /Astrometry: GSC2 Coeff
AMDREX2 =      0.0434354199559 /Astrometry: GSC2 Coeff
AMDREX3 =       -292.435438892 /Astrometry: GSC2 Coeff
AMDREX4 =   4.60919247070E-006 /Astrometry: GSC2 Coeff
AMDREX5 =  -3.21138058537E-006 /Astrometry: GSC2 Coeff
AMDREX6 =   7.23651736725E-006 /Astrometry: GSC2 Coeff
AMDREX7 =       0.000000000000 /Astrometry: GSC2 Coeff
AMDREX8 =       0.000000000000 /Astrometry: GSC2 Coeff
AMDREX9 =       0.000000000000 /Astrometry: GSC2 Coeff
AMDREX10=       0.000000000000 /Astrometry: GSC2 Coeff
AMDREX11=       0.000000000000 /Astrometry: GSC2 Coeff
AMDREX12=       0.000000000000 /Astrometry: GSC2 Coeff
AMDREX13=       0.000000000000 /Astrometry: GSC2 Coeff
AMDREX14=       0.000000000000 /Astrometry: GSC2 Coeff
AMDREX15=       0.000000000000 /Astrometry: GSC2 Coeff
AMDREX16=       0.000000000000 /Astrometry: GSC2 Coeff
```

```
AMDREX17=       0.000000000000 /Astrometry: GSC2 Coeff
AMDREX18=       0.000000000000 /Astrometry: GSC2 Coeff
AMDREX19=       0.000000000000 /Astrometry: GSC2 Coeff
AMDREX20=       0.000000000000 /Astrometry: GSC2 Coeff
AMDREY1 =        67.1522589487 /Astrometry: GSC2 Coeff
AMDREY2 =     -0.0481758265285 /Astrometry: GSC2 Coeff
AMDREY3 =        315.995683716 /Astrometry: GSC2 Coeff
AMDREY4 =  -7.47397531230E-006 /Astrometry: GSC2 Coeff
AMDREY5 =   9.55221105409E-007 /Astrometry: GSC2 Coeff
AMDREY6 =   7.60954485251E-006 /Astrometry: GSC2 Coeff
AMDREY7 =       0.000000000000 /Astrometry: GSC2 Coeff
AMDREY8 =       0.000000000000 /Astrometry: GSC2 Coeff
AMDREY9 =       0.000000000000 /Astrometry: GSC2 Coeff
AMDREY10=       0.000000000000 /Astrometry: GSC2 Coeff
AMDREY11=       0.000000000000 /Astrometry: GSC2 Coeff
AMDREY12=       0.000000000000 /Astrometry: GSC2 Coeff
AMDREY13=       0.000000000000 /Astrometry: GSC2 Coeff
AMDREY14=       0.000000000000 /Astrometry: GSC2 Coeff
AMDREY15=       0.000000000000 /Astrometry: GSC2 Coeff
AMDREY16=       0.000000000000 /Astrometry: GSC2 Coeff
AMDREY17=       0.000000000000 /Astrometry: GSC2 Coeff
AMDREY18=       0.000000000000 /Astrometry: GSC2 Coeff
AMDREY19=       0.000000000000 /Astrometry: GSC2 Coeff
AMDREY20=       0.000000000000 /Astrometry: GSC2 Coeff
ASTRMASK= 'er.mask '           /Astrometry: GSC2 Mask
WCSAXES =                    2 /GetImage: Number WCS axes
WCSNAME = 'DSS               ' /GetImage: Local WCS approximation from full plat
RADESYS = 'ICRS              ' /GetImage: GSC-II calibration using ICRS system
CTYPE1  = 'RA---TAN          ' /GetImage: RA-Gnomic projection
CRPIX1  =           446.000000 /GetImage: X reference pixel
CRVAL1  =            85.274970 /GetImage: RA of reference pixel
CUNIT1  = 'deg               ' /GetImage: degrees
CTYPE2  = 'DEC--TAN          ' /GetImage: Dec-Gnomic projection
CRPIX2  =           447.000000 /GetImage: Y reference pixel
CRVAL2  =            -2.458265 /GetImage: Dec of reference pixel
CUNIT2  = 'deg               ' /Getimage: degrees
CD1_1   =        -0.0002802651 /GetImage: rotation matrix coefficient
CD1_2   =         0.0000003159 /GetImage: rotation matrix coefficient
CD2_1   =         0.0000002767 /GetImage: rotation matrix coefficient
CD2_2   =         0.0002798187 /GetImage: rotation matrix coefficient
OBJECT  = 'data              ' /GetImage: Requested Object Name
DATAMIN =                 3759 /GetImage: Minimum returned pixel value
DATAMAX =                22918 /GetImage: Maximum returned pixel value
OBJCTRA = '05 41 06.000      ' /GetImage: Requested Right Ascension (J2000)
OBJCTDEC= '-02 27 30.00      ' /GetImage: Requested Declination (J2000)
OBJCTX  =             12682.48 /GetImage: Requested X on plate (pixels)
OBJCTY  =             20411.37 /GetImage: Requested Y on plate (pixels)
```

```
OBSERVER= 'Jashanpreet'
CREDIT  = 'NASA    '
```

[6]:
```
image_data = hdu_list[0].data
image_data
```

[6]:
```
array([[ 7201,  6642,  6642, …,  9498,  9498, 10057],
       [ 6642,  6363,  6642, …, 10057, 10616, 10616],
       [ 6922,  6642,  6922, …, 10337, 11175, 10616],
       …,
       [ 5412,  5132,  5412, …, 13000, 12580, 12021],
       [ 5796,  5517,  5796, …, 12546, 12546, 11987],
       [ 5796,  5796,  6076, …, 11987, 12546, 12546]], dtype='>i2')
```

Our data is now stored as a 2D numpy array. But how do we know the dimensions of the image? We can look at the shape of the array.

[10]:
```
print(type(image_data))
print(image_data.shape)
```

```
<class 'numpy.ndarray'>
(893, 891)
```

[11]:
```
image_data1 = fits.getdata(image_file)
image_data1
```

[11]:
```
array([[ 7201,  6642,  6642, …,  9498,  9498, 10057],
       [ 6642,  6363,  6642, …, 10057, 10616, 10616],
       [ 6922,  6642,  6922, …, 10337, 11175, 10616],
       …,
       [ 5412,  5132,  5412, …, 13000, 12580, 12021],
       [ 5796,  5517,  5796, …, 12546, 12546, 11987],
       [ 5796,  5796,  6076, …, 11987, 12546, 12546]], dtype='>i2')
```

[12]:
```
np.array_equal(image_data,image_data1)
```

[12]:
```
True
```

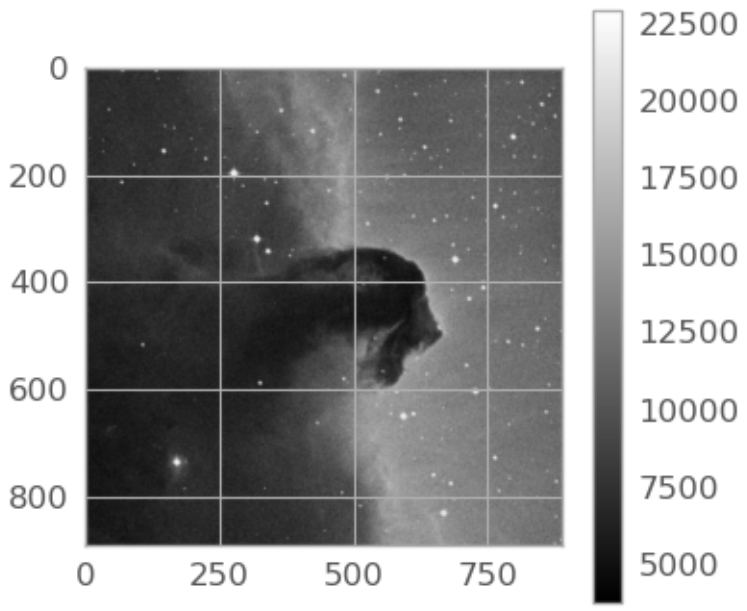
[13]:
```
hdu_list.close()
```

Great! At this point, we can close the FITS file because we've stored everything we wanted to a variable.

Viewing the Image

[15]:
```
plt.figure(figsize=(4,4))
plt.imshow(image_data, cmap='gray')
plt.colorbar()
```

[15]:
```
<matplotlib.colorbar.Colorbar at 0x1c14fe1b690>
```



For more color maps http://wiki.scipy.org/Cookbook/Matplotlib/Show_colormaps OR check chapter Matplotlib of this book.

Let's get some basic statistics about our image:

[17]:
```
print('Min:', np.min(image_data))
print('Max:', np.max(image_data))
print('Mean:', np.mean(image_data))
print('Std deviation:', np.std(image_data))
```

```
Min: 3759
Max: 22918
Mean: 9831.481676287574
Std deviation: 3032.3927542049046
```

Plotting the Data

To make a histogram with matplotlib.pyplot.hist(), we'll need to cast the data from a 2D array to something one dimensional.

In this case, let's use the ndarray.flatten() to return a 1D numpy array.

[18]:
```
image_data.flatten()
```

[18]:
```
array([ 7201,  6642,  6642, …, 11987, 12546, 12546], dtype='>i2')
```

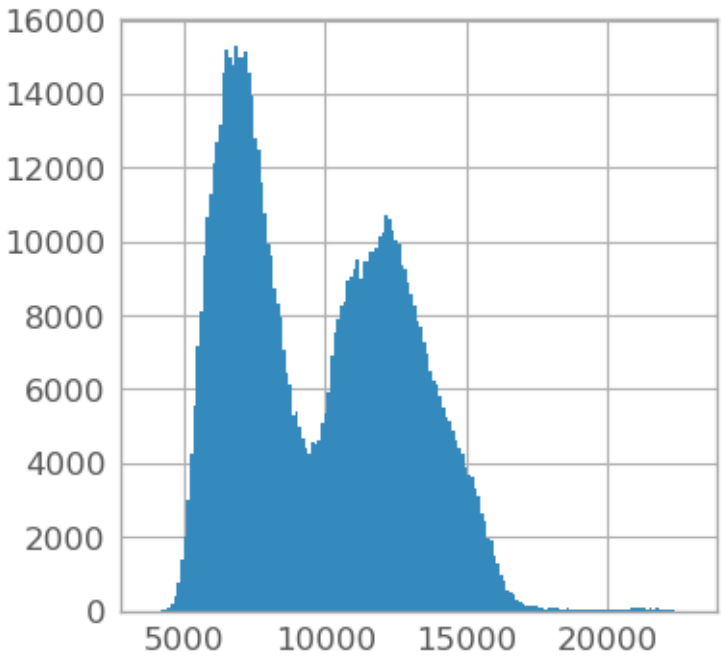
[19]:
```
plt.figure(figsize=(4,4))
print('This is the data stored in numpy array:')
print('and below that its a histogram from the same data')
```

```
plt.hist(image_data.flatten(), bins='auto')
```

This is the data stored in numpy array:
and below that its a histogram from the same data

[19]:
```
(array([1.0000e+00, 0.0000e+00, 1.0000e+00, 3.0000e+00, 1.0000e+01,
        5.4000e+01, 8.0000e+01, 2.0100e+02, 4.1400e+02, 7.4600e+02,
        1.3830e+03, 2.0160e+03, 3.0260e+03, 4.2490e+03, 5.5490e+03,
        7.1710e+03, 8.0890e+03, 9.5940e+03, 1.0663e+04, 1.1307e+04,
        1.2104e+04, 1.2673e+04, 1.3176e+04, 1.4565e+04, 1.5218e+04,
        1.5007e+04, 1.4787e+04, 1.5280e+04, 1.4974e+04, 1.5004e+04,
        1.5140e+04, 1.4559e+04, 1.3922e+04, 1.2821e+04, 1.2500e+04,
        1.1581e+04, 1.0750e+04, 9.9400e+03, 9.6310e+03, 8.7210e+03,
        8.3270e+03, 7.9500e+03, 7.0470e+03, 6.4430e+03, 6.1070e+03,
        5.2880e+03, 5.3750e+03, 4.9840e+03, 4.6900e+03, 4.4270e+03,
        4.2410e+03, 4.5490e+03, 4.5170e+03, 4.6400e+03, 5.0920e+03,
        5.3510e+03, 5.9310e+03, 6.9280e+03, 7.5590e+03, 7.8760e+03,
        8.2450e+03, 8.3850e+03, 8.9250e+03, 9.0700e+03, 9.2730e+03,
        9.4960e+03, 8.9840e+03, 9.4460e+03, 9.4770e+03, 9.7270e+03,
        9.7280e+03, 9.8500e+03, 1.0140e+04, 1.0251e+04, 1.0736e+04,
        1.0588e+04, 1.0320e+04, 1.0011e+04, 9.9160e+03, 9.3340e+03,
        9.2780e+03, 8.9140e+03, 8.5740e+03, 8.2700e+03, 7.8250e+03,
        7.7110e+03, 7.2940e+03, 6.9390e+03, 6.5010e+03, 6.2120e+03,
        6.1100e+03, 5.7930e+03, 5.4990e+03, 5.2420e+03, 5.1190e+03,
        4.8670e+03, 4.6300e+03, 4.4060e+03, 4.2710e+03, 3.8900e+03,
        3.6780e+03, 3.6520e+03, 3.3010e+03, 3.1010e+03, 2.6560e+03,
        2.4170e+03, 1.9660e+03, 1.8920e+03, 1.4730e+03, 1.2600e+03,
        9.5600e+02, 8.0200e+02, 5.7000e+02, 4.9000e+02, 4.3900e+02,
        2.7700e+02, 2.4500e+02, 2.1100e+02, 1.5600e+02, 1.3500e+02,
        1.2400e+02, 1.1400e+02, 8.7000e+01, 6.4000e+01, 5.6000e+01,
        6.3000e+01, 6.1000e+01, 7.4000e+01, 6.2000e+01, 5.0000e+01,
        4.9000e+01, 6.8000e+01, 5.3000e+01, 4.7000e+01, 5.0000e+01,
        4.4000e+01, 5.0000e+01, 5.0000e+01, 3.9000e+01, 3.7000e+01,
        3.4000e+01, 4.4000e+01, 3.8000e+01, 4.3000e+01, 3.9000e+01,
        4.0000e+01, 5.5000e+01, 3.6000e+01, 4.5000e+01, 4.8000e+01,
        5.1000e+01, 6.5000e+01, 7.2000e+01, 8.2000e+01, 7.3000e+01,
        7.4000e+01, 5.8000e+01, 8.2000e+01, 5.5000e+01, 7.2000e+01,
        4.2000e+01, 3.9000e+01, 2.3000e+01, 1.8000e+01, 1.2000e+01,
        8.0000e+00, 7.0000e+00, 6.0000e+00, 0.0000e+00, 4.0000e+00]),
 array([ 3759. ,  3871.7,  3984.4,  4097.1,  4209.8,  4322.5,  4435.2,
         4547.9,  4660.6,  4773.3,  4886. ,  4998.7,  5111.4,  5224.1,
         5336.8,  5449.5,  5562.2,  5674.9,  5787.6,  5900.3,  6013. ,
         6125.7,  6238.4,  6351.1,  6463.8,  6576.5,  6689.2,  6801.9,
         6914.6,  7027.3,  7140. ,  7252.7,  7365.4,  7478.1,  7590.8,
         7703.5,  7816.2,  7928.9,  8041.6,  8154.3,  8267. ,  8379.7,
         8492.4,  8605.1,  8717.8,  8830.5,  8943.2,  9055.9,  9168.6,
         9281.3,  9394. ,  9506.7,  9619.4,  9732.1,  9844.8,  9957.5,
```

```
        10070.2, 10182.9, 10295.6, 10408.3, 10521. , 10633.7, 10746.4,
        10859.1, 10971.8, 11084.5, 11197.2, 11309.9, 11422.6, 11535.3,
        11648. , 11760.7, 11873.4, 11986.1, 12098.8, 12211.5, 12324.2,
        12436.9, 12549.6, 12662.3, 12775. , 12887.7, 13000.4, 13113.1,
        13225.8, 13338.5, 13451.2, 13563.9, 13676.6, 13789.3, 13902. ,
        14014.7, 14127.4, 14240.1, 14352.8, 14465.5, 14578.2, 14690.9,
        14803.6, 14916.3, 15029. , 15141.7, 15254.4, 15367.1, 15479.8,
        15592.5, 15705.2, 15817.9, 15930.6, 16043.3, 16156. , 16268.7,
        16381.4, 16494.1, 16606.8, 16719.5, 16832.2, 16944.9, 17057.6,
        17170.3, 17283. , 17395.7, 17508.4, 17621.1, 17733.8, 17846.5,
        17959.2, 18071.9, 18184.6, 18297.3, 18410. , 18522.7, 18635.4,
        18748.1, 18860.8, 18973.5, 19086.2, 19198.9, 19311.6, 19424.3,
        19537. , 19649.7, 19762.4, 19875.1, 19987.8, 20100.5, 20213.2,
        20325.9, 20438.6, 20551.3, 20664. , 20776.7, 20889.4, 21002.1,
        21114.8, 21227.5, 21340.2, 21452.9, 21565.6, 21678.3, 21791. ,
        21903.7, 22016.4, 22129.1, 22241.8, 22354.5, 22467.2, 22579.9,
        22692.6, 22805.3, 22918. ]),
<BarContainer object of 170 artists>)
```

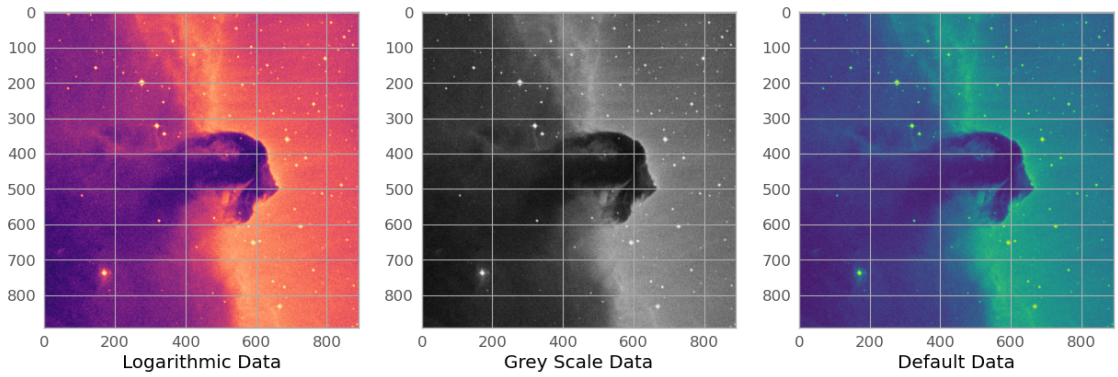


Logarithmic Analysis

What if we want to use a logarithmic color scale? To do so, we'll need to load the LogNorm object from matplotlib.

[7]:
```
from matplotlib.colors import LogNorm

plt.figure(figsize=(15,15))

plt.subplot(1,3,1)
plt.xlabel("Logarithmic Data")
plt.imshow(image_data, cmap='magma', norm=LogNorm())

plt.subplot(1,3,2)
plt.xlabel("Grey Scale Data")
plt.imshow(image_data, cmap='gray')

plt.subplot(1,3,3)
plt.xlabel("Default Data")
plt.imshow(image_data)
```

[7]:
```
<matplotlib.image.AxesImage at 0x1fc6d9cac10>
```



3. SDSS Data Analysis and AGNs Classification

i. SDSS QUERY

In this chapter, we embark on an exploration of the cosmos using the powerful tools provided by the Sloan Digital Sky Survey (SDSS) and the Astroquery library. Our journey begins with an introduction to the SDSS, a monumental project that has meticulously mapped and cataloged the celestial landscape, capturing data from millions of stars, galaxies, and quasars.

Utilizing the Astroquery library, we delve into the SDSS Science Archive Server (SAS), accessing its vast repository of astronomical data. Through the SAS, we conduct precise queries and perform cross-matching exercises, leveraging the wealth of information stored within this comprehensive database.

Our focus extends beyond mere data retrieval; we seek to unravel the mysteries of the universe by employing advanced techniques. We meticulously craft visualizations, bringing forth the spectra of our extracted data. Through these plotted spectra, we aim to unravel cosmic signatures, decode the nature of celestial bodies, and reveal the hidden stories written within the light emitted from distant corners of our universe.

As we navigate this celestial tapestry, interpreting the whispers of light from distant galaxies, and uncovering the secrets held within the vast cosmic expanse, we will draw insights from the SDSS Data Release 18 (DR18). This latest release encapsulates the cutting-edge discoveries and observations made by the SDSS, offering a rich repository of data and insights. For further exploration and reference, the SDSS DR18 can be accessed at dr18.sdss.org.

[75]:
```
import pandas as pd
import matplotlib.pyplot as plt
from astroquery.sdss import SDSS
from astroquery.sdss import SDSS
```

Querying on Python

1. Select galaxies and quasars with redshifts between 0.05 and 0.3 and signalto-noise ratios greater than 35 near the H line. Ensure that lines [O III] 5007, H 4863, and H 4341 are present in emission and that the FWHM of H is greater than 1000 km/s. For each selected spectrum, find the flux ratios of [OIII]/H , H /H , and [O III]/H , as well as the equivalent width and flux of H , redshift, and extinction correction: E(B-V) of type SFD (tip: the last one find in galSpecInfo table).

We will use the following query using https://skyserver.sdss.org/dr18/SearchTools/sql:

```
s.plate, s.mjd, s.fiberid,
s.z, g.subclass, g.e_bv_sfd,
l.h_beta_flux, l.h_beta_eqw,
(l.oiii_5007_flux/l.h_beta_flux) as oiii_h_beta_flux_ratio,
(l.oiii_5007_flux/l.h_gamma_flux) as oiii_h_gamma_flux_ratio,
(l.h_beta_flux/l.h_gamma_flux) as h_beta_h_gamma_flux_ratio

FROM SpecObjAll AS s
JOIN GalSpecInfo AS g ON s.specobjid = g.specobjid
```

```
JOIN GalSpecLine AS l ON s.specobjid = l.specobjid

WHERE
(s.class = "QSO" OR s.class = "GALAXY")
AND s.z BETWEEN 0.05 AND 0.6
AND s.snmedian_g > 35
AND l.h_gamma_eqw < 0
AND l.h_beta_eqw < 0
AND l.oiii_5007_eqw < 0
AND l.h_gamma_flux <> 0
AND l.h_beta_flux <> 0
AND l.oiii_5007_flux <> 0
AND l.sigma_balmer * 2.35 > 1000
```

You can paste this query not the site. Alternatively, this could be done with python as follows.

Resulting table in *420primary.csv*

[78]:
```
with open('query.txt', 'r') as file:
    query = file.read().replace('\n', '') # input is a single string


res = SDSS.query_sql(query)
res # output is an astropy table
```

[78]:
```
<Table length=420>
plate  mjd  fiberid … oiii_h_gamma_flux_ratio h_beta_h_gamma_flux_ratio
int32 int32  int32  …         float64                   float64
----- ----- ------- … ----------------------- -------------------------
  390 51900     587 …                1.645157                  1.677394
  355 51788     414 …                1.879482                  2.642485
  519 52283     280 …                2.025708                  1.964605
 1233 52734     312 …                2.169284                  1.449437
  494 51915     124 …                3.919306                  2.127565
 2647 54495     378 …                1.048939                 0.8033115
  607 52368     581 …                1.061532                  2.359785
 2227 53820     252 …                3.758329                  1.402163
 2156 54525     122 …                1.029048                  2.071693
  394 51913     404 …               0.7865114                 0.8310005
  …   …      … …                     …                         …
 1058 52520     379 …               0.6717112                  2.142452
 1272 52989     539 …               0.9215587                 0.9550459
 1601 53115     118 …                 1.33615                  2.155445
 2524 54568     468 …                1.159231                  2.006942
  721 52228     454 …               0.9839695                  2.428448
 2341 53738     456 …                0.848797                 0.9598599
  464 51908     576 …               0.4592269                  1.133334
 2646 54479     204 …               0.2995124                  1.913876
```

```
2755 54507     376 …          1.177436          1.660927
2612 54480     330 …         0.3271222          2.025746
```

2. How many objects have you found? Which one from the conditions in WHERE is narrowing the results most severely?

condition	count
all	420
z removed	1621
sn removed	35502

The signal-to-noise ratio condition has the most impact on the result.

[79]:
```
conditions = [
    "s.class = 'QSO' OR s.class = 'GALAXY'",
    "s.z BETWEEN 0.05 AND 0.6",
    "s.snmedian_g > 35",
    # ... add other conditions here
]

for condition in conditions:
    # Remove one condition at a time and re-run the query
    modified_query = query.replace(f"AND {condition}", "")
    modified_res = SDSS.query_sql(modified_query)
    num_objects = len(modified_res)
    print(f"After removing condition: {condition}, Number of objects found:",␣
 ↪num_objects)
```

```
After removing condition: s.class = 'QSO' OR s.class = 'GALAXY', Number of
objects found: 420
After removing condition: s.z BETWEEN 0.05 AND 0.6, Number of objects found:
1621
After removing condition: s.snmedian_g > 35, Number of objects found: 35502
```

3. Find out if there is some of the Subclass AGN objects, with the same conditions under 1.

[80]:
```
# Define the SQL query with an additional condition for subclass AGN
query = '''
SELECT
    s.plate,
    s.mjd,
    s.fiberid,
    s.z,
    g.subclass,
    g.e_bv_sfd,
    l.h_beta_flux,
    l.h_beta_eqw,
    (l.oiii_5007_flux / l.h_beta_flux) as oiii_h_beta_flux_ratio,
```

```
    (l.oiii_5007_flux / l.h_gamma_flux) as oiii_h_gamma_flux_ratio,
    (l.h_beta_flux / l.h_gamma_flux) as h_beta_h_gamma_flux_ratio
FROM
    SpecObjAll AS s
JOIN
    GalSpecInfo AS g ON s.specobjid = g.specobjid
JOIN
    GalSpecLine AS l ON s.specobjid = l.specobjid
WHERE
    (s.class = 'QSO' OR s.class = 'GALAXY')
    AND s.z BETWEEN 0.05 AND 0.6
    AND s.snmedian_g > 35
    AND l.h_gamma_eqw < 0
    AND l.h_beta_eqw < 0
    AND l.oiii_5007_eqw < 0
    AND l.h_gamma_flux <> 0
    AND l.h_beta_flux <> 0
    AND l.oiii_5007_flux <> 0
    AND l.sigma_balmer * 2.35 > 1000
    AND g.subclass LIKE '%AGN%'  -- Adding condition for AGN subclass
'''

# Execute the modified query to retrieve data for subclass AGN objects
result_AGN = SDSS.query_sql(query)
number_AGN = len(result_AGN)
print("Yes, there are AGNs Broadline =",number_AGN)
```

```
Yes, there are AGNs Broadline = 14
```

Crossmatching Data

4. Using the problem solution under 1 and the list of objects (287-plate-mjdfiber.txt) submit the SQL query via CrossID. (TIP: you will need to alter the SQL code prepared under 1 to fit requirements of CrossID. Follow the comments you get and be patient)

refer data.txt in material (https://astrodingra/academics/material.html)

[81]:
```
#data ~ data.txt
data = """
( 1, 1949, 53433, 472),( 2, 1273, 52993, 348),( 3, 2030, 53499, 201),
( 4, 2520, 54584, 442),( 5, 1940, 53383, 407),( 6, 561, 52295, 618),
( 7, 1603, 53119, 558),( 8, 453, 51915, 212),( 9, 1427, 52996, 606),
( 10, 464, 51908, 576),( 11, 634, 52164, 430),( 12, 2020, 53431, 614),
( 13, 2007, 53474, 171),( 14, 916, 52378, 434),( 15, 603, 52056, 415),
( 16, 2089, 53498, 385),( 17, 625, 52145, 288),( 18, 436, 51883, 16),
( 19, 999, 52636, 334),( 20, 1683, 53436, 16),( 21, 2425, 54139, 442),
( 22, 433, 51873, 422),( 23, 465, 51910, 603),( 24, 513, 51989, 340),
( 25, 366, 52017, 584),( 26, 329, 52056, 577),( 27, 564, 52224, 368),
```

```
( 28, 564, 52224, 471),( 29, 753, 52233, 455),( 30, 755, 52235, 225),
( 31, 1052, 52466, 370),( 32, 738, 52521, 95),( 33, 993, 52710, 147),
( 34, 914, 52721, 74),( 35, 1351, 52790, 428),( 36, 1342, 52793, 157),
( 37, 1864, 53313, 353),( 38, 2491, 53855, 252),( 39, 848, 52669, 418),
( 40, 555, 52266, 74),( 41, 1371, 52821, 377),( 42, 1844, 54138, 112),
( 43, 1417, 53141, 441),( 44, 2783, 54524, 432),( 45, 2641, 54230, 570),
( 46, 490, 51929, 126),( 47, 1796, 53884, 591),( 48, 2509, 54180, 244),
( 49, 2647, 54495, 160),( 50, 2488, 54149, 521),( 51, 1186, 52646, 98),
( 52, 2164, 53886, 503),( 53, 640, 52178, 513),( 54, 2646, 54479, 204),
( 55, 2583, 54095, 615),( 56, 2501, 54084, 120),( 57, 2128, 53800, 415),
( 58, 2518, 54243, 374),( 59, 413, 51929, 598),( 60, 837, 52642, 298),
( 61, 2645, 54477, 133),( 62, 2353, 53794, 600),( 63, 1843, 53816, 584),
( 64, 2970, 54589, 373),( 65, 967, 52636, 44),( 66, 767, 52252, 300),
( 67, 593, 52026, 111),( 68, 984, 52442, 383),( 69, 1429, 52990, 584),
( 70, 2769, 54527, 354),( 71, 2532, 54589, 90),( 72, 2166, 54232, 30),
( 73, 2281, 53711, 431),( 74, 2420, 54086, 240),( 75, 597, 52059, 337),
( 76, 1598, 53033, 291),( 77, 1237, 52762, 497),( 78, 593, 52026, 170),
( 79, 656, 52148, 282),( 80, 461, 51910, 74),( 81, 1750, 53358, 564),
( 82, 764, 52238, 53),( 83, 2512, 53877, 582),( 84, 469, 51913, 238),
( 85, 892, 52378, 395),( 86, 268, 51633, 235),( 87, 1793, 53883, 573),
( 88, 1598, 53033, 604),( 89, 2595, 54207, 541),( 90, 1817, 53851, 69),
( 91, 349, 51699, 613),( 92, 2440, 53818, 539),( 93, 1713, 53827, 63),
( 94, 567, 52252, 558),( 95, 1690, 53475, 634),( 96, 385, 51877, 61),
( 97, 2351, 53772, 99),( 98, 1230, 52672, 503),( 99, 1431, 52992, 572),
( 100, 2971, 54590, 326),( 101, 1575, 53493, 484),( 102, 2764, 54535, 53),
( 103, 1773, 53112, 301),( 104, 498, 51984, 104),( 105, 411, 51817, 381),
( 106, 2034, 53466, 230),( 107, 637, 52174, 259),( 108, 2529, 54585, 300),
( 109, 600, 52317, 490),( 110, 1223, 52781, 625),( 111, 551, 51993, 179),
( 112, 2478, 54097, 377),( 113, 721, 52228, 454),( 114, 1676, 53147, 55),
( 115, 1626, 53472, 449),( 116, 614, 53437, 452),( 117, 1754, 53385, 324),
( 118, 2364, 53737, 469),( 119, 2606, 54154, 614),( 120, 1775, 53847, 420),
( 121, 1791, 54266, 46),( 122, 2080, 53350, 309),( 123, 2294, 53733, 607),
( 124, 1838, 53467, 451),( 125, 2351, 53772, 616),( 126, 1167, 52738, 476),
( 127, 1695, 53473, 612),( 128, 1989, 53772, 559),( 129, 1667, 53430, 114),
( 130, 2885, 54497, 561),( 131, 2008, 53473, 615),( 132, 2347, 53757, 326),
( 133, 2341, 53738, 66),( 134, 1378, 53061, 413),( 135, 1949, 53433, 437),
( 136, 1779, 53089, 106),( 137, 1201, 52674, 225),( 138, 752, 52251, 20),
( 139, 511, 52636, 567),( 140, 2424, 54448, 109),( 141, 1428, 52998, 558),
( 142, 2101, 53858, 348),( 143, 666, 52149, 24),( 144, 628, 52083, 461),
( 145, 793, 52370, 549),( 146, 1714, 53521, 11),( 147, 1872, 53386, 371),
( 148, 1725, 54266, 540),( 149, 1944, 53385, 412),( 150, 2522, 54570, 131),
( 151, 664, 52174, 455),( 152, 939, 52636, 172),( 153, 2347, 53757, 151),
( 154, 836, 52376, 262),( 155, 1329, 52767, 542),( 156, 773, 52376, 629),
( 157, 1433, 53035, 148),( 158, 2376, 53770, 497),( 159, 2592, 54178, 153),
( 160, 1455, 53089, 584),( 161, 899, 52620, 626),( 162, 723, 52201, 500),
( 163, 896, 52592, 63),( 164, 597, 52059, 520),( 165, 554, 52000, 553),
( 166, 1453, 53084, 10),( 167, 1865, 53312, 572),( 168, 942, 52703, 488),
```

```
( 169, 897, 52605, 242),( 170, 2222, 53799, 388),( 171, 1847, 54176, 630),
( 172, 1807, 54175, 9),( 173, 1364, 53061, 185),( 174, 795, 52378, 381),
( 175, 2517, 54567, 624),( 176, 1372, 53062, 488),( 177, 1367, 53083, 629),
( 178, 1773, 53112, 405),( 179, 2583, 54095, 462),( 180, 932, 52620, 110),
( 181, 2237, 53828, 93),( 182, 1604, 53078, 281),( 183, 2173, 53874, 354),
( 184, 408, 51821, 611),( 185, 768, 52281, 85),( 186, 1694, 53472, 540),
( 187, 1394, 53108, 44),( 188, 411, 51817, 519),( 189, 1941, 53386, 545),
( 190, 653, 52145, 185),( 191, 2106, 53714, 130),( 192, 994, 52725, 86),
( 193, 2152, 53874, 108),( 194, 1714, 53521, 571),( 195, 1378, 53061, 488),
( 196, 1606, 53055, 166),( 197, 1195, 52724, 98),( 198, 1854, 53566, 151),
( 199, 2772, 54529, 38),( 200, 650, 52143, 126),( 201, 1843, 53816, 8),
( 202, 1619, 53084, 60),( 203, 722, 52224, 281),( 204, 2023, 53851, 65),
( 205, 526, 52312, 378),( 206, 2003, 53442, 501),( 207, 510, 52381, 509),
( 208, 2365, 53739, 591),( 209, 2617, 54502, 287),( 210, 666, 52149, 496),
( 211, 2526, 54582, 404),( 212, 409, 51871, 213),( 213, 1803, 54152, 323),
( 214, 1795, 54507, 509),( 215, 2362, 53759, 295),( 216, 1926, 53317, 378),
( 217, 860, 52319, 583),( 218, 2216, 53795, 298),( 219, 1810, 53794, 504),
( 220, 2508, 53875, 353),( 221, 1744, 53055, 387),( 222, 1271, 52974, 462),
( 223, 1214, 52731, 160),( 224, 2493, 54115, 278),( 225, 1735, 53035, 322),
( 226, 1218, 52709, 201),( 227, 1457, 53116, 421),( 228, 607, 52368, 581),
( 229, 767, 52252, 453),( 230, 2102, 54115, 18),( 231, 2352, 53770, 242),
( 232, 507, 52353, 161),( 233, 2613, 54481, 620),( 234, 976, 52413, 574),
( 235, 814, 52443, 467),( 236, 2265, 53674, 87),( 237, 1645, 53172, 526),
( 238, 2615, 54483, 404),( 239, 987, 52523, 157),( 240, 792, 52353, 116),
( 241, 2744, 54272, 535),( 242, 1728, 53228, 218),( 243, 1716, 53827, 350),
( 244, 2007, 53474, 226),( 245, 827, 52312, 621),( 246, 882, 52370, 376),
( 247, 1199, 52703, 596),( 248, 1402, 52872, 594),( 249, 1446, 53080, 135),
( 250, 2615, 54483, 257),( 251, 594, 52045, 73),( 252, 2276, 53712, 508),
( 253, 1845, 54144, 637),( 254, 1362, 53050, 119),( 255, 596, 52370, 483),
( 256, 2151, 54523, 527),( 257, 1215, 52725, 150),( 258, 1224, 52765, 379),
(259, 1843, 53816, 502),( 260, 856, 52339, 50),( 261, 1440, 53084, 137),
( 262, 941, 52709, 616),( 263, 2233, 53845, 196),( 264, 1592, 52990, 18),
( 265, 2795, 54563, 140),( 266, 2428, 53801, 356),( 267, 489, 51930, 402),
( 268, 795, 52378, 528),( 269, 1214, 52731, 77),( 270, 1005, 52703, 518),
( 271, 402, 51793, 479),( 272, 2242, 54153, 199),( 273, 1004, 52723, 280),
( 274, 1310, 53033, 559),( 275, 1380, 53084, 58),( 276, 1054, 52516, 309),
( 277, 2492, 54178, 562),( 278, 2955, 54562, 608),( 279, 353, 51703, 546),
( 280, 644, 52173, 413),( 281, 1432, 53003, 91),( 282, 741, 52261, 116),
( 283, 1171, 52753, 416),( 284, 2774, 54534, 51),( 285, 2485, 54176, 231),
( 286, 2292, 53713, 348)
"""
last_comma_index = data.rfind(",")
data_with_line_break = data[:last_comma_index] + ",\n" + data[last_comma_index␣
↪+ 1:]
print(data_with_line_break)
```

```
( 1, 1949, 53433, 472),( 2, 1273, 52993, 348),( 3, 2030, 53499, 201),( 4, 2520,
```

54584, 442),(5, 1940, 53383, 407),(6, 561, 52295, 618),(7, 1603, 53119,
558),(8, 453, 51915, 212),(9, 1427, 52996, 606),(10, 464, 51908, 576),(11,
634, 52164, 430),(12, 2020, 53431, 614),(13, 2007, 53474, 171),(14, 916,
52378, 434),(15, 603, 52056, 415),(16, 2089, 53498, 385),(17, 625, 52145,
288),(18, 436, 51883, 16),(19, 999, 52636, 334),(20, 1683, 53436, 16),(21,
2425, 54139, 442),(22, 433, 51873, 422),(23, 465, 51910, 603),(24, 513,
51989, 340),(25, 366, 52017, 584),(26, 329, 52056, 577),(27, 564, 52224,
368),(28, 564, 52224, 471),(29, 753, 52233, 455),(30, 755, 52235, 225),(31,
1052, 52466, 370),(32, 738, 52521, 95),(33, 993, 52710, 147),(34, 914, 52721,
74),(35, 1351, 52790, 428),(36, 1342, 52793, 157),(37, 1864, 53313, 353),(
38, 2491, 53855, 252),(39, 848, 52669, 418),(40, 555, 52266, 74),(41, 1371,
52821, 377),(42, 1844, 54138, 112),(43, 1417, 53141, 441),(44, 2783, 54524,
432),(45, 2641, 54230, 570),(46, 490, 51929, 126),(47, 1796, 53884, 591),(
48, 2509, 54180, 244),(49, 2647, 54495, 160),(50, 2488, 54149, 521),(51,
1186, 52646, 98),(52, 2164, 53886, 503),(53, 640, 52178, 513),(54, 2646,
54479, 204),(55, 2583, 54095, 615),(56, 2501, 54084, 120),(57, 2128, 53800,
415),(58, 2518, 54243, 374),(59, 413, 51929, 598),(60, 837, 52642, 298),(61,
2645, 54477, 133),(62, 2353, 53794, 600),(63, 1843, 53816, 584),(64, 2970,
54589, 373),(65, 967, 52636, 44),(66, 767, 52252, 300),(67, 593, 52026,
111),(68, 984, 52442, 383),(69, 1429, 52990, 584),(70, 2769, 54527, 354),(
71, 2532, 54589, 90),(72, 2166, 54232, 30),(73, 2281, 53711, 431),(74, 2420,
54086, 240),(75, 597, 52059, 337),(76, 1598, 53033, 291),(77, 1237, 52762,
497),(78, 593, 52026, 170),(79, 656, 52148, 282),(80, 461, 51910, 74),(81,
1750, 53358, 564),(82, 764, 52238, 53),(83, 2512, 53877, 582),(84, 469,
51913, 238),(85, 892, 52378, 395),(86, 268, 51633, 235),(87, 1793, 53883,
573),(88, 1598, 53033, 604),(89, 2595, 54207, 541),(90, 1817, 53851, 69),(
91, 349, 51699, 613),(92, 2440, 53818, 539),(93, 1713, 53827, 63),(94, 567,
52252, 558),(95, 1690, 53475, 634),(96, 385, 51877, 61),(97, 2351, 53772,
99),(98, 1230, 52672, 503),(99, 1431, 52992, 572),(100, 2971, 54590, 326),(
101, 1575, 53493, 484),(102, 2764, 54535, 53),(103, 1773, 53112, 301),(104,
498, 51984, 104),(105, 411, 51817, 381),(106, 2034, 53466, 230),(107, 637,
52174, 259),(108, 2529, 54585, 300),(109, 600, 52317, 490),(110, 1223, 52781,
625),(111, 551, 51993, 179),(112, 2478, 54097, 377),(113, 721, 52228, 454),(
114, 1676, 53147, 55),(115, 1626, 53472, 449),(116, 614, 53437, 452),(117,
1754, 53385, 324),(118, 2364, 53737, 469),(119, 2606, 54154, 614),(120, 1775,
53847, 420),(121, 1791, 54266, 46),(122, 2080, 53350, 309),(123, 2294, 53733,
607),(124, 1838, 53467, 451),(125, 2351, 53772, 616),(126, 1167, 52738,
476),(127, 1695, 53473, 612),(128, 1989, 53772, 559),(129, 1667, 53430,
114),(130, 2885, 54497, 561),(131, 2008, 53473, 615),(132, 2347, 53757,
326),(133, 2341, 53738, 66),(134, 1378, 53061, 413),(135, 1949, 53433, 437),(
136, 1779, 53089, 106),(137, 1201, 52674, 225),(138, 752, 52251, 20),(139,
511, 52636, 567),(140, 2424, 54448, 109),(141, 1428, 52998, 558),(142, 2101,
53858, 348),(143, 666, 52149, 24),(144, 628, 52083, 461),(145, 793, 52370,
549),(146, 1714, 53521, 11),(147, 1872, 53386, 371),(148, 1725, 54266, 540),(
149, 1944, 53385, 412),(150, 2522, 54570, 131),(151, 664, 52174, 455),(152,
939, 52636, 172),(153, 2347, 53757, 151),(154, 836, 52376, 262),(155, 1329,
52767, 542),(156, 773, 52376, 629),(157, 1433, 53035, 148),(158, 2376, 53770,
497),(159, 2592, 54178, 153),(160, 1455, 53089, 584),(161, 899, 52620, 626),(

```
162, 723, 52201, 500),( 163, 896, 52592, 63),( 164, 597, 52059, 520),( 165, 554,
52000, 553),( 166, 1453, 53084, 10),( 167, 1865, 53312, 572),( 168, 942, 52703,
488),( 169, 897, 52605, 242),( 170, 2222, 53799, 388),( 171, 1847, 54176, 630),(
172, 1807, 54175, 9),( 173, 1364, 53061, 185),( 174, 795, 52378, 381),( 175,
2517, 54567, 624),( 176, 1372, 53062, 488),( 177, 1367, 53083, 629),( 178, 1773,
53112, 405),( 179, 2583, 54095, 462),( 180, 932, 52620, 110),( 181, 2237, 53828,
93),( 182, 1604, 53078, 281),( 183, 2173, 53874, 354),( 184, 408, 51821, 611),(
185, 768, 52281, 85),( 186, 1694, 53472, 540),( 187, 1394, 53108, 44),( 188,
411, 51817, 519),( 189, 1941, 53386, 545),( 190, 653, 52145, 185),( 191, 2106,
53714, 130),( 192, 994, 52725, 86),( 193, 2152, 53874, 108),( 194, 1714, 53521,
571),( 195, 1378, 53061, 488),( 196, 1606, 53055, 166),( 197, 1195, 52724, 98),(
198, 1854, 53566, 151),( 199, 2772, 54529, 38),( 200, 650, 52143, 126),( 201,
1843, 53816, 8),( 202, 1619, 53084, 60),( 203, 722, 52224, 281),( 204, 2023,
53851, 65),( 205, 526, 52312, 378),( 206, 2003, 53442, 501),( 207, 510, 52381,
509),( 208, 2365, 53739, 591),( 209, 2617, 54502, 287),( 210, 666, 52149, 496),(
211, 2526, 54582, 404),( 212, 409, 51871, 213),( 213, 1803, 54152, 323),( 214,
1795, 54507, 509),( 215, 2362, 53759, 295),( 216, 1926, 53317, 378),( 217, 860,
52319, 583),( 218, 2216, 53795, 298),( 219, 1810, 53794, 504),( 220, 2508,
53875, 353),( 221, 1744, 53055, 387),( 222, 1271, 52974, 462),( 223, 1214,
52731, 160),( 224, 2493, 54115, 278),( 225, 1735, 53035, 322),( 226, 1218,
52709, 201),( 227, 1457, 53116, 421),( 228, 607, 52368, 581),( 229, 767, 52252,
453),( 230, 2102, 54115, 18),( 231, 2352, 53770, 242),( 232, 507, 52353, 161),(
233, 2613, 54481, 620),( 234, 976, 52413, 574),( 235, 814, 52443, 467),( 236,
2265, 53674, 87),( 237, 1645, 53172, 526),( 238, 2615, 54483, 404),( 239, 987,
52523, 157),( 240, 792, 52353, 116),( 241, 2744, 54272, 535),( 242, 1728, 53228,
218),( 243, 1716, 53827, 350),( 244, 2007, 53474, 226),( 245, 827, 52312, 621),(
246, 882, 52370, 376),( 247, 1199, 52703, 596),( 248, 1402, 52872, 594),( 249,
1446, 53080, 135),( 250, 2615, 54483, 257),( 251, 594, 52045, 73),( 252, 2276,
53712, 508),( 253, 1845, 54144, 637),( 254, 1362, 53050, 119),( 255, 596, 52370,
483),( 256, 2151, 54523, 527),( 257, 1215, 52725, 150),( 258, 1224, 52765,
379),( 259, 1843, 53816, 502),( 260, 856, 52339, 50),( 261, 1440, 53084, 137),(
262, 941, 52709, 616),( 263, 2233, 53845, 196),( 264, 1592, 52990, 18),( 265,
2795, 54563, 140),( 266, 2428, 53801, 356),( 267, 489, 51930, 402),( 268, 795,
52378, 528),( 269, 1214, 52731, 77),( 270, 1005, 52703, 518),( 271, 402, 51793,
479),( 272, 2242, 54153, 199),( 273, 1004, 52723, 280),( 274, 1310, 53033,
559),( 275, 1380, 53084, 58),( 276, 1054, 52516, 309),( 277, 2492, 54178, 562),(
278, 2955, 54562, 608),( 279, 353, 51703, 546),( 280, 644, 52173, 413),( 281,
1432, 53003, 91),( 282, 741, 52261, 116),( 283, 1171, 52753, 416),( 284, 2774,
54534, 51),( 285, 2485, 54176, 231),( 286, 2292, 53713,
 348)


SELECT
s.plate, s.mjd, s.fiberid,
s.z, g.subclass, g.e_bv_sfd,
l.h_beta_flux, l.h_beta_eqw,
(l.oiii_5007_flux/l.h_beta_flux) as oiii_h_beta_flux_ratio,
(l.oiii_5007_flux/l.h_gamma_flux) as oiii_h_gamma_flux_ratio,
```

```
(l.h_beta_flux/l.h_gamma_flux) as h_beta_h_gamma_flux_ratio

FROM #upload u
JOIN
SpecObjAll AS s
ON (s.plate=u.up_plate AND s.mjd=u.up_mjd AND s.fiberID=u.up_fiber)
JOIN GalSpecInfo AS g ON s.specobjid = g.specobjid
JOIN GalSpecLine AS l ON s.specobjid = l.specobjid

WHERE
(s.class = "QSO" OR s.class = "GALAXY")
AND s.z BETWEEN 0.05 AND 0.6
AND s.snmedian_g > 35
AND l.h_gamma_eqw < 0
AND l.h_beta_eqw < 0
AND l.oiii_5007_eqw < 0
AND l.h_gamma_flux > 0
AND l.h_beta_flux > 0
AND l.oiii_5007_flux > 0
AND (l.sigma_balmer * 2.35) > 1000
```

This produced 47 rows. Results in *47crossmatch.csv*

Rechecked with pandas:

[82]:
```
result = pd.read_csv("47crossmatch.csv", sep=',', header=0, comment='#')
result.head()
```

[82]:

	plate	mjd	fiberid	z	subclass	e_bv_sfd	h_beta_flux	\
0	1224	52765	379	0.522838	BROADLINE	0.040022	1333.4460	
1	411	51817	381	0.532471	BROADLINE	0.088108	1313.9030	
2	2347	53757	151	0.511335	BROADLINE	0.026489	1452.4410	
3	1716	53827	350	0.500819	BROADLINE	0.033267	798.7177	
4	2520	54584	442	0.435857	BROADLINE	0.053350	178.4678	

	h_beta_eqw	oiii_h_beta_flux_ratio	oiii_h_gamma_flux_ratio	\
0	-12.693350	0.339881	0.930526	
1	-10.747670	0.420567	1.289833	
2	-16.707320	0.689349	1.492386	
3	-9.304626	0.902137	1.717440	
4	-11.221260	3.537334	6.079030	

	h_beta_h_gamma_flux_ratio
0	2.737802
1	3.066892
2	2.164921
3	1.903746
4	1.718534

5. Check the spectra of found objects, download some of them using wget.
6. BONUS: read downloaded fits files and plot the spectra using Python.

Since the SAS server was down, I explored other options to download spectra. Here I am employing astroquery to plot and understand the results.

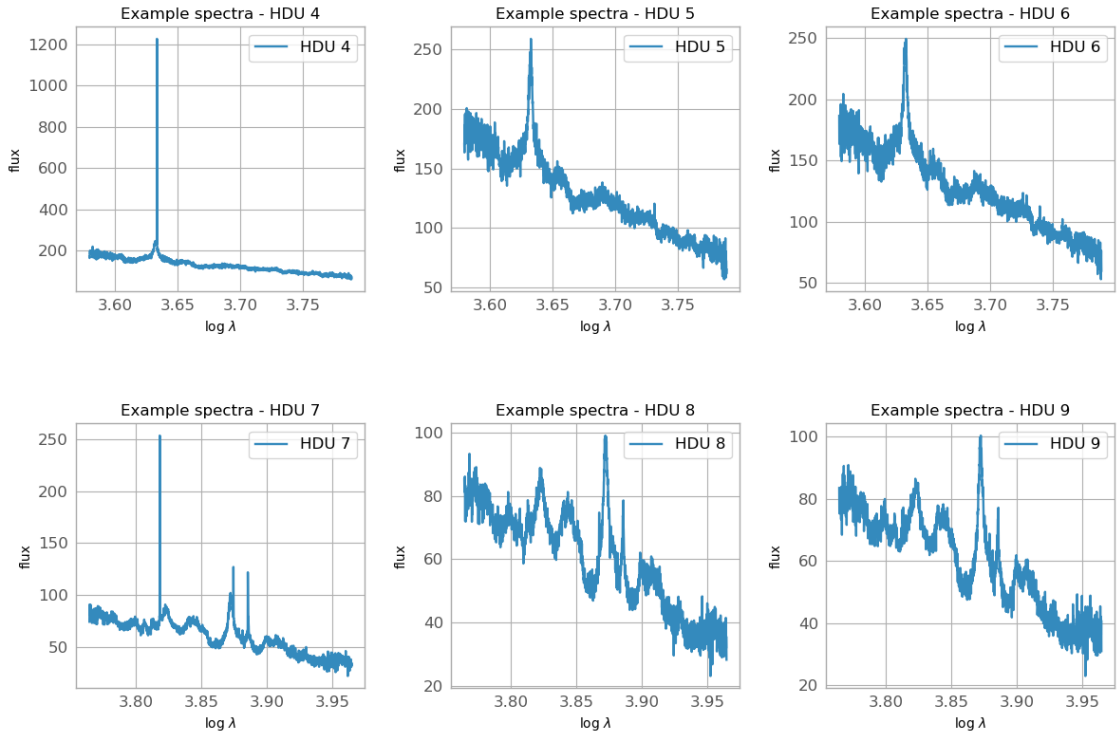
[83]:
```
sp = SDSS.get_spectra(plate=result.loc[1,"plate"], mjd=result.loc[1,"mjd"],␣
 ↪fiberID=result.loc[1,"fiberid"]) # list of HDUs
sp[0].info()
```

```
Filename: (No file associated with this HDUList)
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU     138   ()
  1  COADD         1 BinTableHDU     26   3848R x 8C   [E, E, E, J, J, E, E, E]
  2  SPECOBJ       1 BinTableHDU    262   1R x 126C   [6A, 4A, 16A, 23A, 16A,
8A, E, E, E, J, E, E, J, B, B, B, B, B, B, J, 22A, 19A, 19A, 22A, 19A, I, 3A,
3A, 1A, J, D, D, D, E, E, 19A, 8A, J, J, J, J, K, K, J, J, J, J, J, J, K, K, K,
K, I, J, J, J, J, 5J, D, D, 6A, 21A, E, E, E, J, E, 24A, 10J, J, 10E, E, E, E,
E, E, E, J, E, E, E, J, E, 5E, E, 10E, 10E, 10E, 5E, 5E, 5E, 5E, 5E, J, J, E, E,
E, E, E, E, 25A, 21A, 10A, E, E, E, E, E, E, E, E, J, E, E, J, 1A, 1A, E, E, J,
J, 1A, 5E, 5E]
  3  SPZLINE       1 BinTableHDU     48   29R x 19C   [J, J, J, 13A, D, E, E, E,
E, E, E, E, E, E, E, J, J, E, E]
  4  B2-00006798-00006802-00006803    1 BinTableHDU    145   2047R x 7C   [E, E,
E, J, E, E, E]
  5  B2-00006799-00006802-00006803    1 BinTableHDU    145   2047R x 7C   [E, E,
E, J, E, E, E]
  6  B2-00006800-00006802-00006803    1 BinTableHDU    145   2047R x 7C   [E, E,
E, J, E, E, E]
  7  R2-00006798-00006802-00006803    1 BinTableHDU    145   2044R x 7C   [E, E,
E, J, E, E, E]
  8  R2-00006799-00006802-00006803    1 BinTableHDU    145   2044R x 7C   [E, E,
E, J, E, E, E]
  9  R2-00006800-00006802-00006803    1 BinTableHDU    145   2044R x 7C   [E, E,
E, J, E, E, E]
```

[84]:
```
# Set the figure size to make subplots larger
plt.figure(figsize=(10, 8))  # Adjust the size as needed

# Since sp is your list of spectra
num_spectra = 6  # Number of spectra in the range

# Create subplots with appropriate size and spacing
fig, axs = plt.subplots(2, 3, figsize=(12, 8))  # 2 rows, 3 columns

for i in range(4, 10):
    spectrum = sp[0][i]
```

```
    row = (i - 4) // 3
    col = (i - 4) % 3

    axs[row, col].plot(spectrum.data['loglam'], spectrum.data['flux'],␣
 ↪label='HDU ' + str(i))
    axs[row, col].set_xlabel(r'log $\lambda$', fontsize=10)  # Adjust label␣
 ↪font size
    axs[row, col].set_ylabel('flux', fontsize=10)  # Adjust label font size
    axs[row, col].set_title(f"Example spectra - HDU {i}", fontsize=12)  #␣
 ↪Adjust title font size
    axs[row, col].legend()

# Adjust layout and spacing
plt.tight_layout()
plt.subplots_adjust(hspace=0.5, wspace=0.3)  # Adjust horizontal and vertical␣
 ↪spacing

plt.show()
```

```
<Figure size 1000x800 with 0 Axes>
```



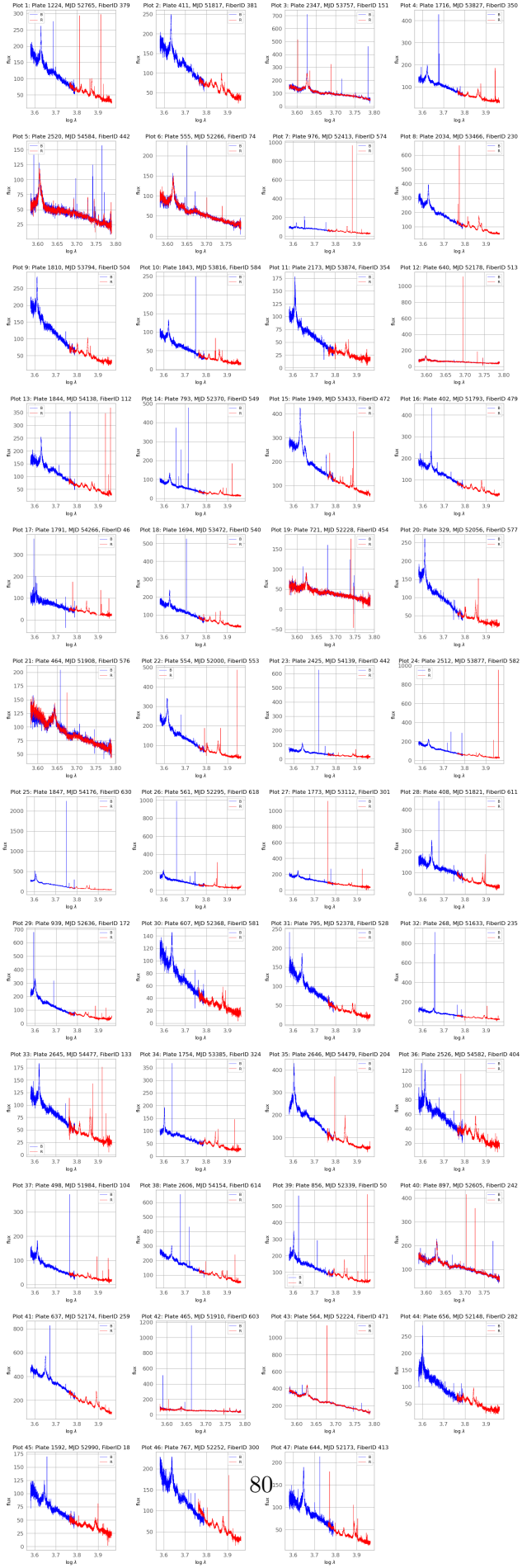
Observations: - HDUs 4-6 are the bluer wavelengths and 7-9 correspond to the redder wavelengths - HDUs 6 and 9 are the reduced results

Final 47 Spectra of Crossmatched Data

A convenient way to study the results is by using subplots.

[86]:
```
import matplotlib.pyplot as plt

# Assuming result is a DataFrame containing plate, mjd, and fiberID information

num_plots = len(result)

# Calculate rows and columns for subplots
num_cols = 4  # Adjust the number of plots per row
num_rows = (num_plots + num_cols - 1) // num_cols

fig, axs = plt.subplots(num_rows, num_cols, figsize=(16, 4 * num_rows))  #␣
 ↪Adjust height based on the number of rows

for i in range(num_plots):
    plate = result.loc[i, "plate"]
    mjd = result.loc[i, "mjd"]
    fiberID = result.loc[i, "fiberid"]
    sp = SDSS.get_spectra(plate=plate, mjd=mjd, fiberID=fiberID)
    spectrum_b = sp[0][6]
    spectrum_r = sp[0][9]

    row = i // num_cols
    col = i % num_cols

    axs[row, col].plot(spectrum_b.data['loglam'], spectrum_b.data['flux'],␣
 ↪label='B', c='b', linewidth=0.5)  # Adjust line width
    axs[row, col].plot(spectrum_r.data['loglam'], spectrum_r.data['flux'],␣
 ↪label='R', c='r', linewidth=0.5)
    axs[row, col].set_xlabel(r'log $\lambda$', fontsize=10)  # Adjust label␣
 ↪font size
    axs[row, col].set_ylabel('flux', fontsize=10)  # Adjust label font size
    axs[row, col].set_title(f"Plot {i+1}: Plate {plate}, MJD {mjd}, FiberID␣
 ↪{fiberID}", fontsize=12)  # Add plot number
    axs[row, col].legend(fontsize=8)  # Adjust legend font size

# Remove empty subplots
for i in range(num_plots, num_rows * num_cols):
    row = i // num_cols
    col = i % num_cols
    fig.delaxes(axs[row, col])


# Adjust layout and spacing
plt.tight_layout(pad=2.0)# Increase spacing between subplots
```

```
# Save the plot as final47.png
plt.savefig('final47.png')
plt.show()
```


Plot 1: Plate 1224, MJD 52765, FiberID 379
Plot 2: Plate 411, MJD 51817, FiberID 381
Plot 3: Plate 2347, MJD 53757, FiberID 151
Plot 4: Plate 1716, MJD 53827, FiberID 350
Plot 5: Plate 2520, MJD 54584, FiberID 442
Plot 6: Plate 555, MJD 52266, FiberID 74
Plot 7: Plate 976, MJD 52413, FiberID 574
Plot 8: Plate 2034, MJD 53466, FiberID 230
Plot 9: Plate 1810, MJD 53794, FiberID 504
Plot 10: Plate 1843, MJD 53816, FiberID 584
Plot 11: Plate 2173, MJD 53874, FiberID 354
Plot 12: Plate 640, MJD 52178, FiberID 513
Plot 13: Plate 1844, MJD 54138, FiberID 112
Plot 14: Plate 793, MJD 52370, FiberID 549
Plot 15: Plate 1949, MJD 53433, FiberID 472
Plot 16: Plate 402, MJD 51793, FiberID 479
Plot 17: Plate 1791, MJD 54266, FiberID 46
Plot 18: Plate 1694, MJD 53472, FiberID 540
Plot 19: Plate 721, MJD 52228, FiberID 454
Plot 20: Plate 329, MJD 52056, FiberID 577
Plot 21: Plate 464, MJD 51908, FiberID 576
Plot 22: Plate 554, MJD 52000, FiberID 553
Plot 23: Plate 2425, MJD 54139, FiberID 442
Plot 24: Plate 2512, MJD 53877, FiberID 582
Plot 25: Plate 1847, MJD 54176, FiberID 630
Plot 26: Plate 561, MJD 52295, FiberID 618
Plot 27: Plate 1773, MJD 53112, FiberID 301
Plot 28: Plate 408, MJD 51821, FiberID 611
Plot 29: Plate 939, MJD 52636, FiberID 172
Plot 30: Plate 607, MJD 52368, FiberID 581
Plot 31: Plate 795, MJD 52378, FiberID 528
Plot 32: Plate 268, MJD 51633, FiberID 235
Plot 33: Plate 2645, MJD 54477, FiberID 133
Plot 34: Plate 1754, MJD 53385, FiberID 324
Plot 35: Plate 2646, MJD 54479, FiberID 204
Plot 36: Plate 2526, MJD 54582, FiberID 404
Plot 37: Plate 498, MJD 51984, FiberID 104
Plot 38: Plate 2606, MJD 54154, FiberID 614
Plot 39: Plate 856, MJD 52339, FiberID 50
Plot 40: Plate 897, MJD 52605, FiberID 242
Plot 41: Plate 637, MJD 52174, FiberID 259
Plot 42: Plate 465, MJD 51910, FiberID 603
Plot 43: Plate 564, MJD 52224, FiberID 471
Plot 44: Plate 656, MJD 52148, FiberID 282
Plot 45: Plate 1592, MJD 52990, FiberID 18
Plot 46: Plate 767, MJD 52252, FiberID 300
Plot 47: Plate 644, MJD 52173, FiberID 413

Or, if you prefer, you can plot the graphs using the code provided separately

[85]:
```
for i in range(len(result)):
    print("# ", i+1)
    plate=result.loc[i,"plate"]
    mjd=result.loc[i,"mjd"]
    fiberID=result.loc[i,"fiberid"]
    sp = SDSS.get_spectra(plate=plate, mjd=mjd, fiberID=fiberID)
    spectrum_b = sp[0][6]
    spectrum_r = sp[0][9]
    plt.plot(spectrum_b.data['loglam'], spectrum_b.data['flux'], label='B',␣
 ↪c='b')
    plt.plot(spectrum_r.data['loglam'], spectrum_r.data['flux'], label='R',␣
 ↪c='r')
    plt.xlabel(r'log $\lambda$')
    plt.ylabel('flux')
    plt.title(f"Plate: {plate} MJD: {mjd} FiberID: {fiberID}")
    plt.legend()
    plt.show()
```

```
#  1
```

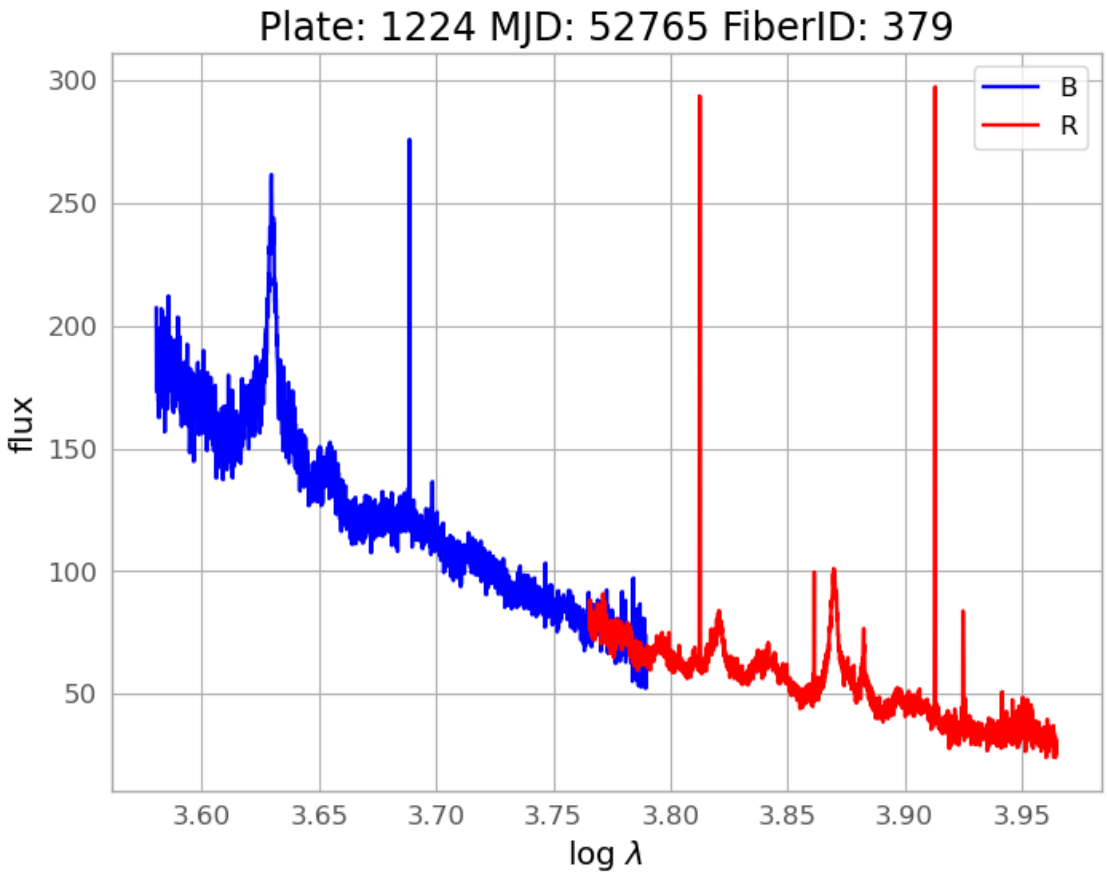


Plate: 1224 MJD: 52765 FiberID: 379
B
R
flux
log λ
300
250
200
150
100
50
3.60
3.65
3.70
3.75
3.80
3.85
3.90
3.95

2

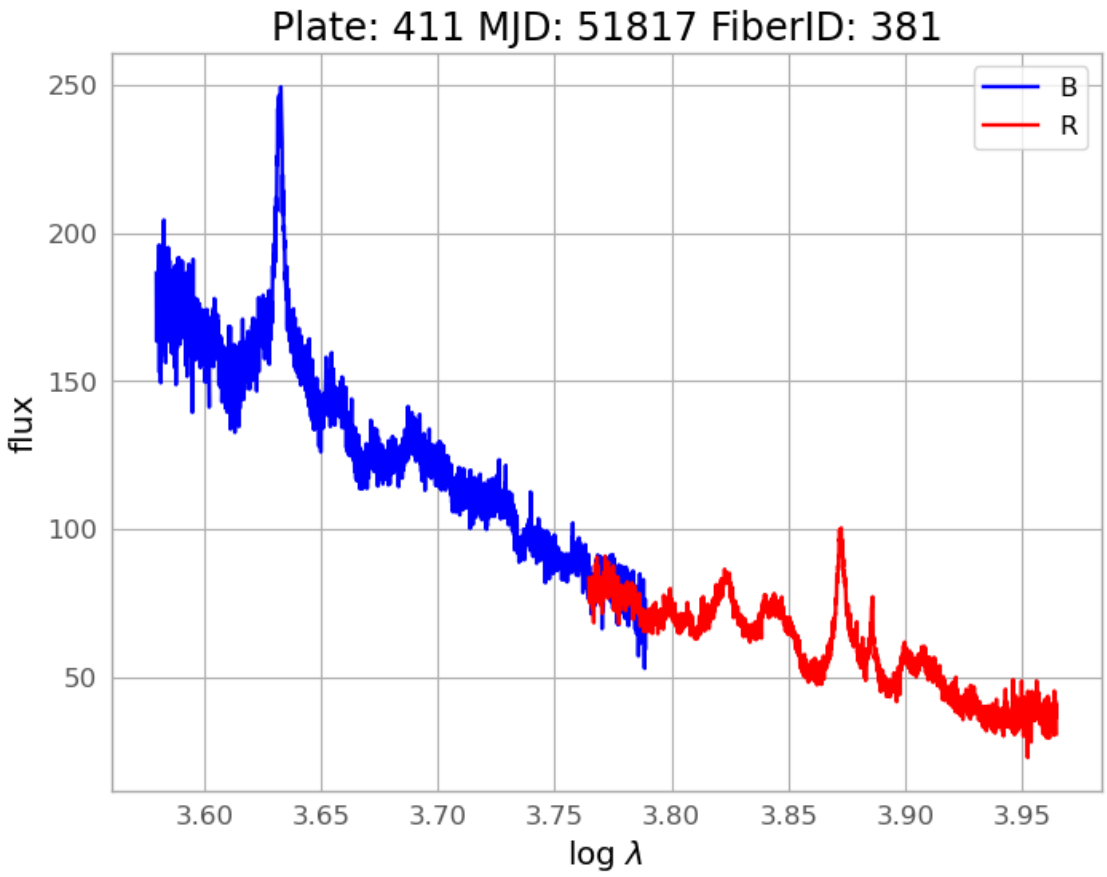


Plate: 411 MJD: 51817 FiberID: 381
B
R
flux
log λ
250
200
150
100
50
3.60
3.65
3.70
3.75
3.80
3.85
3.90
3.95

3

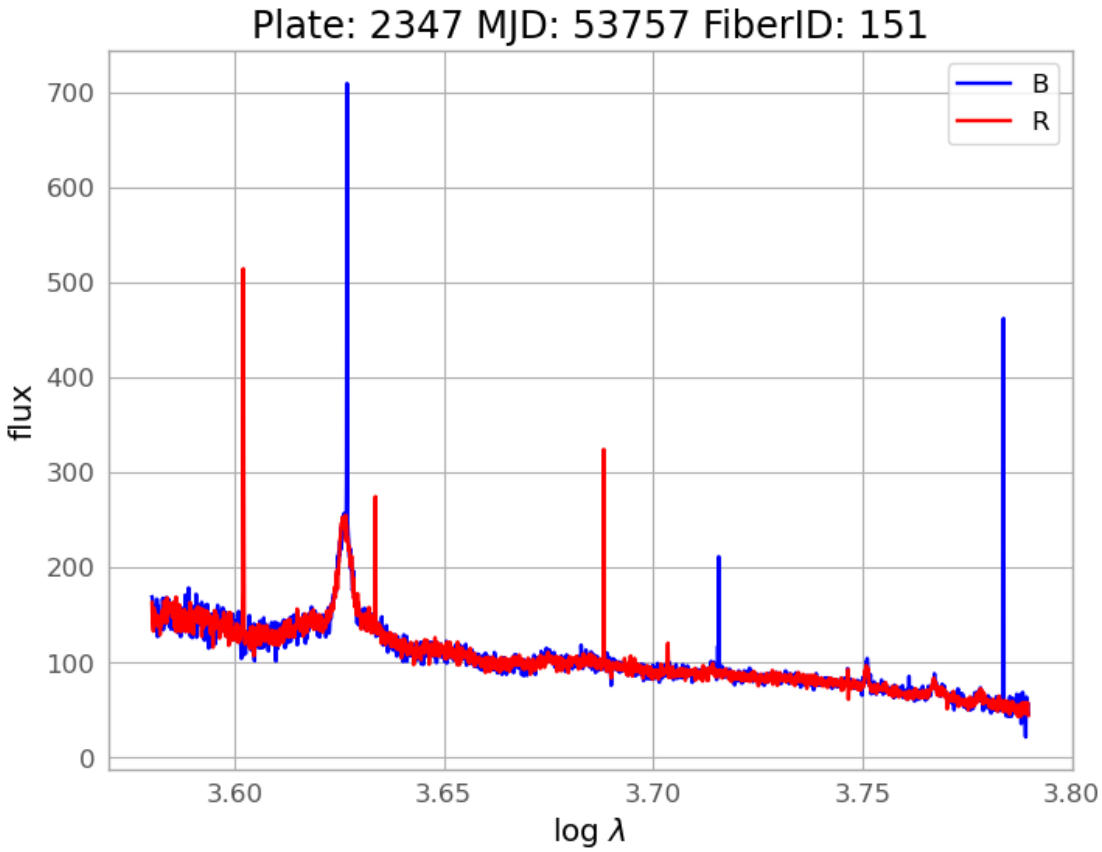


Plate: 2347 MJD: 53757 FiberID: 151
B
R
flux
log λ
0
100
200
300
400
500
600
700
3.60
3.65
3.70
3.75
3.80

```
#  4
```

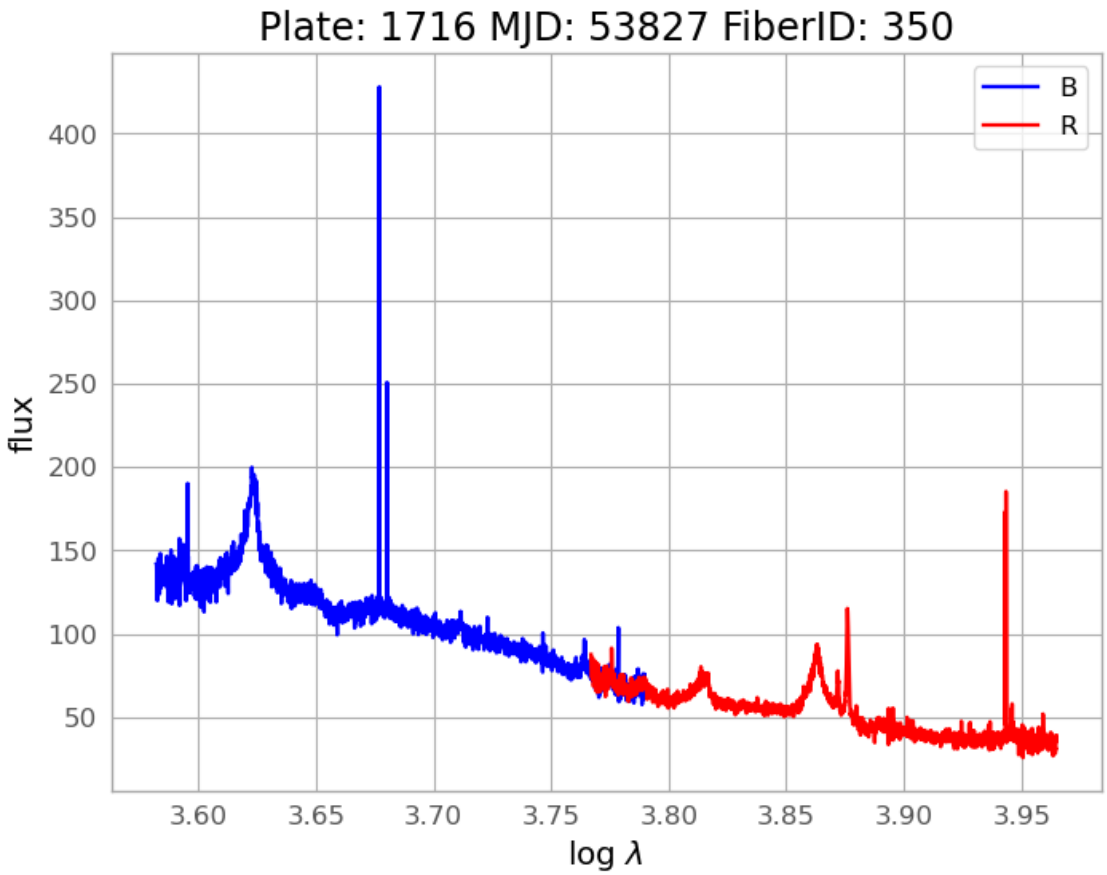


Plate: 1716 MJD: 53827 FiberID: 350
B
R
flux
400
350
300
250
200
150
100
50
3.60
3.65
3.70
3.75
3.80
3.85
3.90
3.95
log λ

5

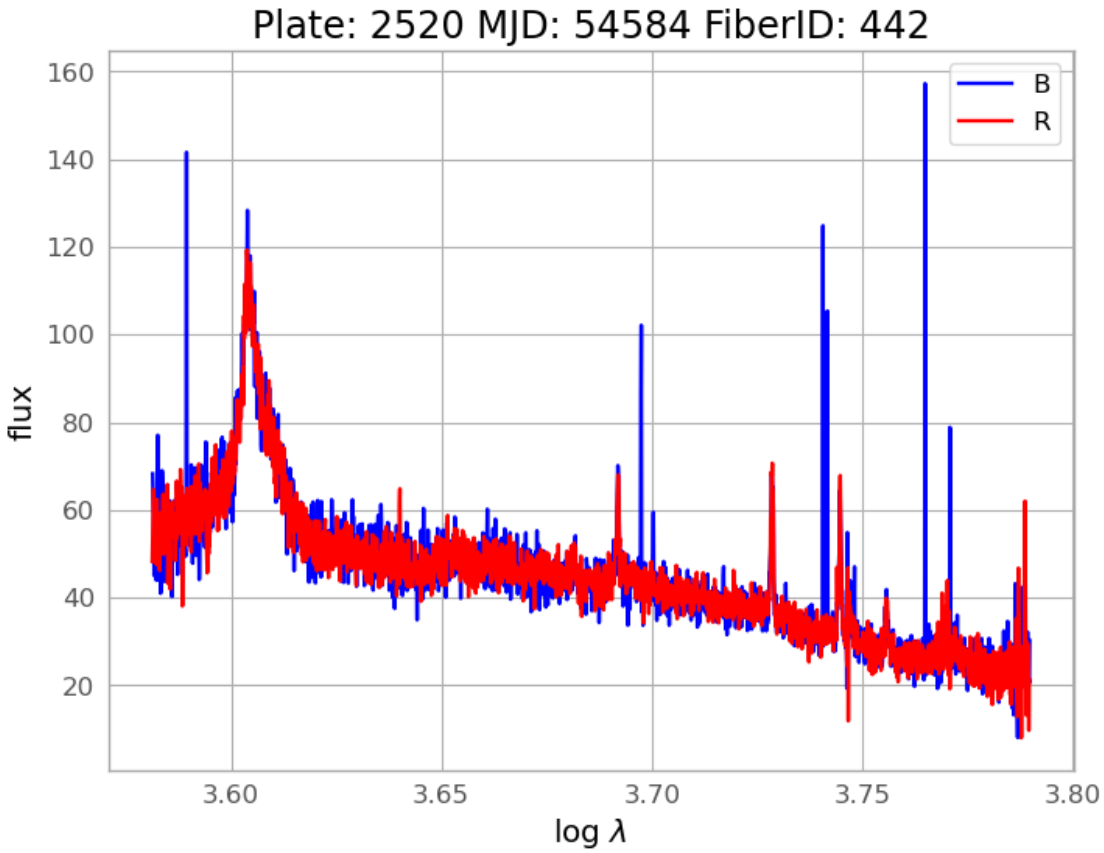


Plate: 2520 MJD: 54584 FiberID: 442
B
R
flux
log λ
160
140
120
100
80
60
40
20
3.60
3.65
3.70
3.75
3.80

6

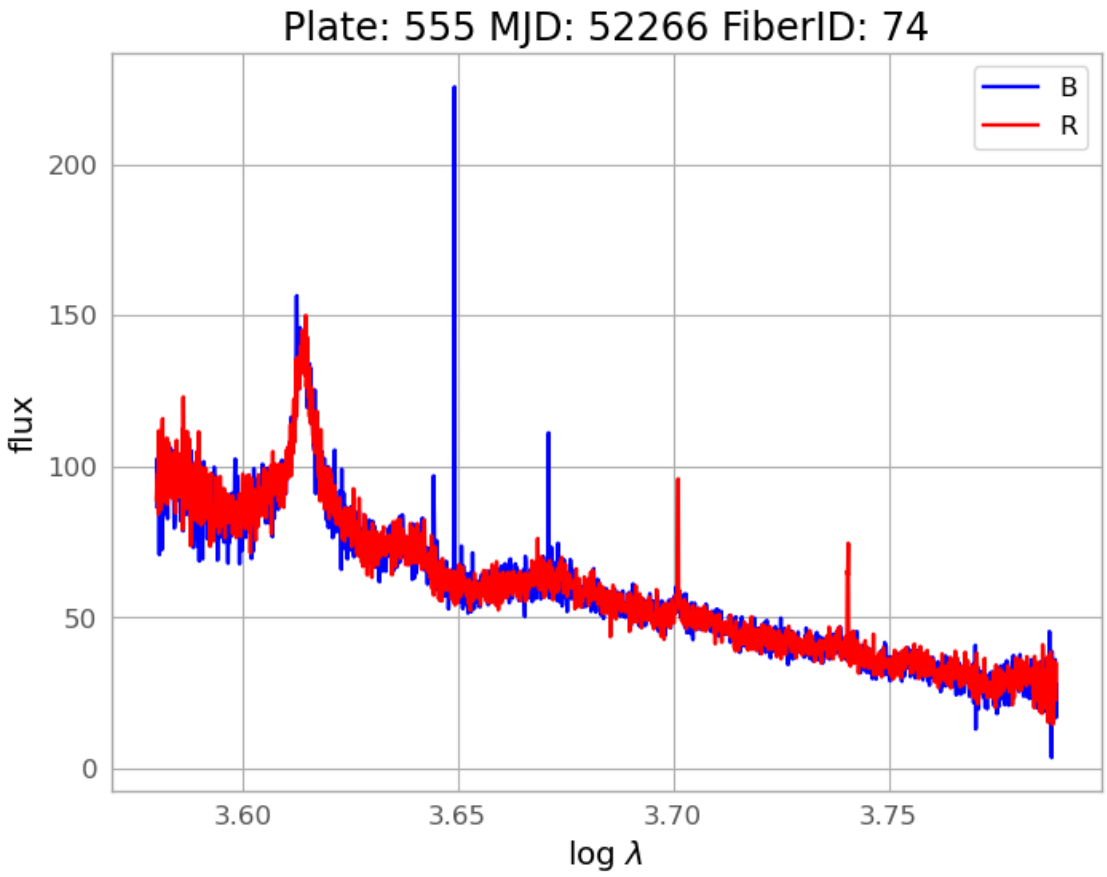


Plate: 555 MJD: 52266 FiberID: 74
B
R
200
150
100
50
0
flux
3.60
3.65
3.70
3.75
log λ

7

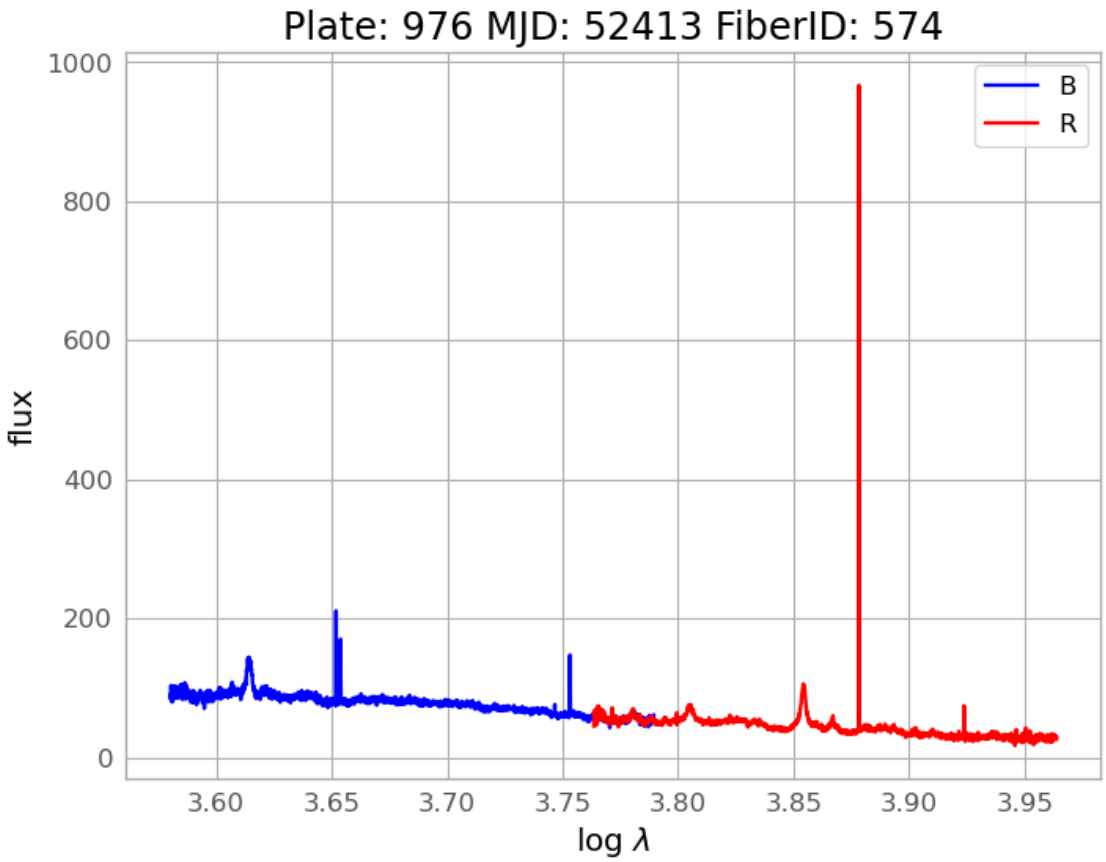


Plate: 976 MJD: 52413 FiberID: 574
B
R
flux
log λ
1000
800
600
400
200
0
3.60
3.65
3.70
3.75
3.80
3.85
3.90
3.95

```
#  8
```

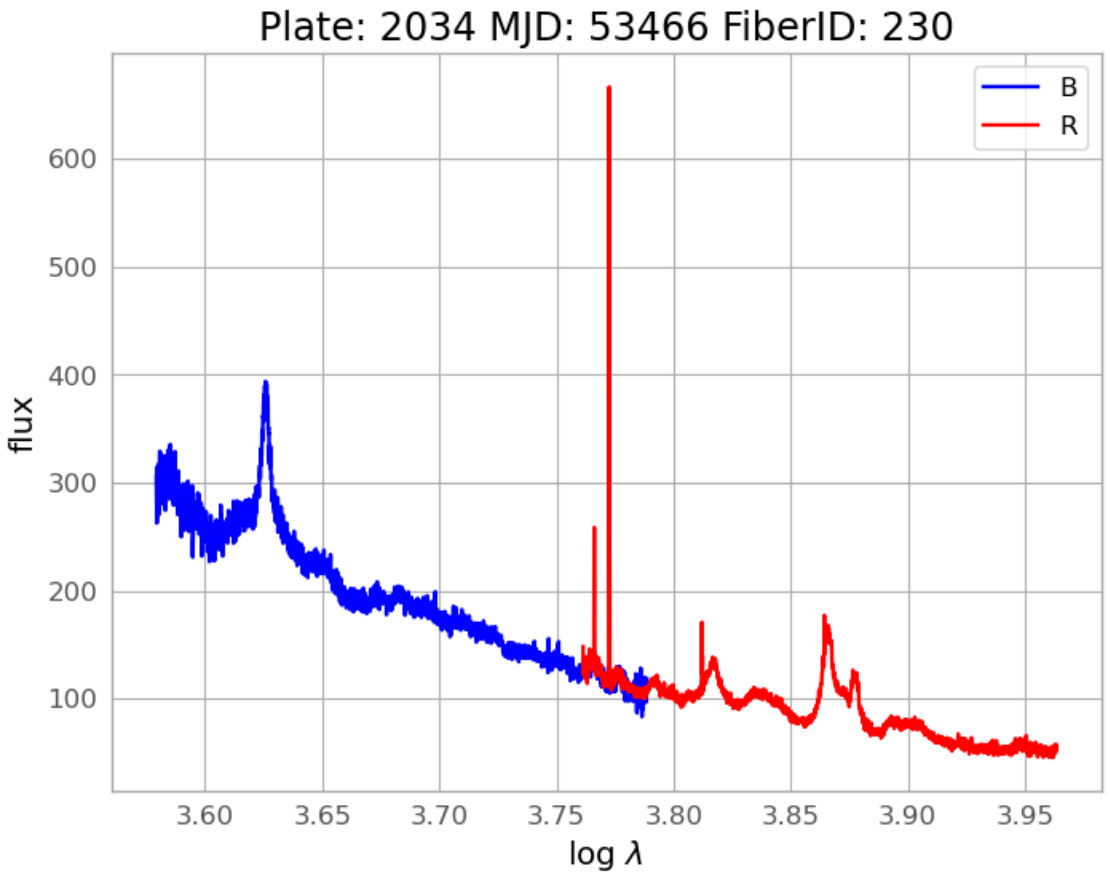


Plate: 2034 MJD: 53466 FiberID: 230
B
R
flux
600
500
400
300
200
100
3.60
3.65
3.70
3.75
3.80
3.85
3.90
3.95
log λ

9

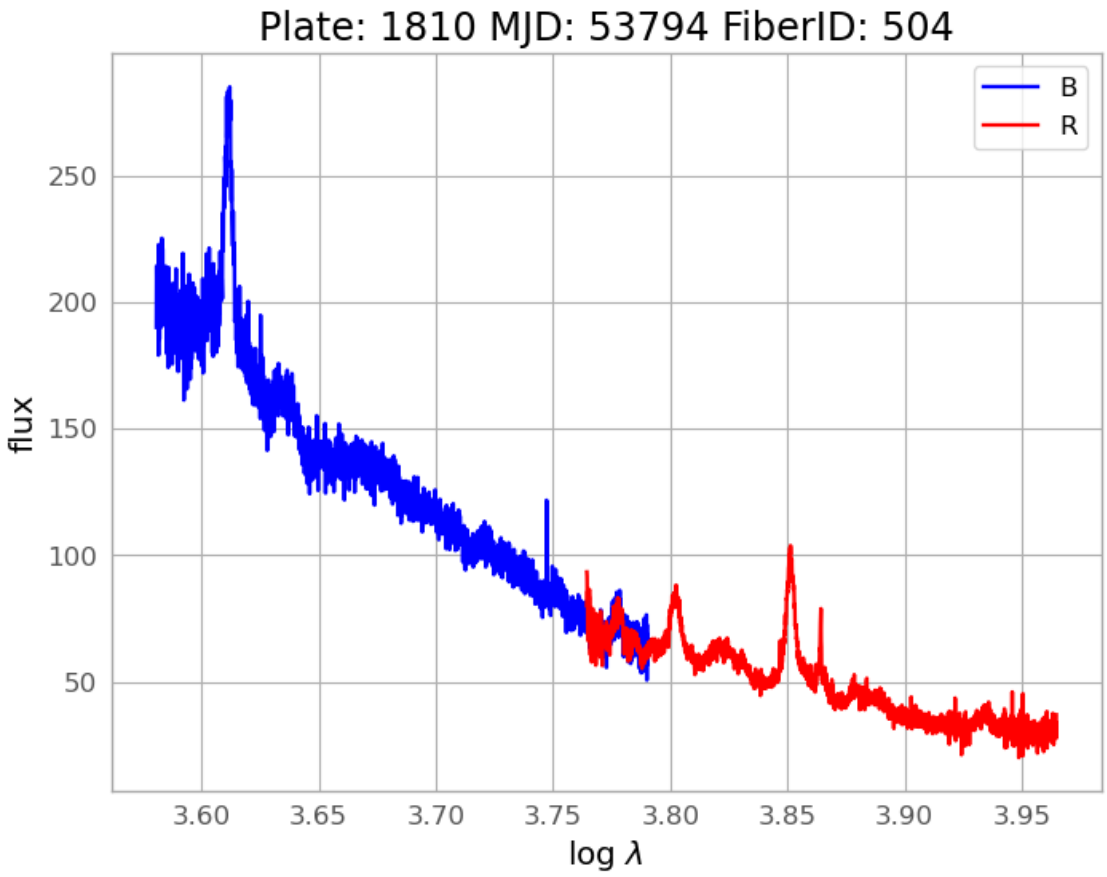


Plate: 1810 MJD: 53794 FiberID: 504
B
R
flux
250
200
150
100
50
3.60
3.65
3.70
3.75
3.80
3.85
3.90
3.95
log λ

```
#  10
```

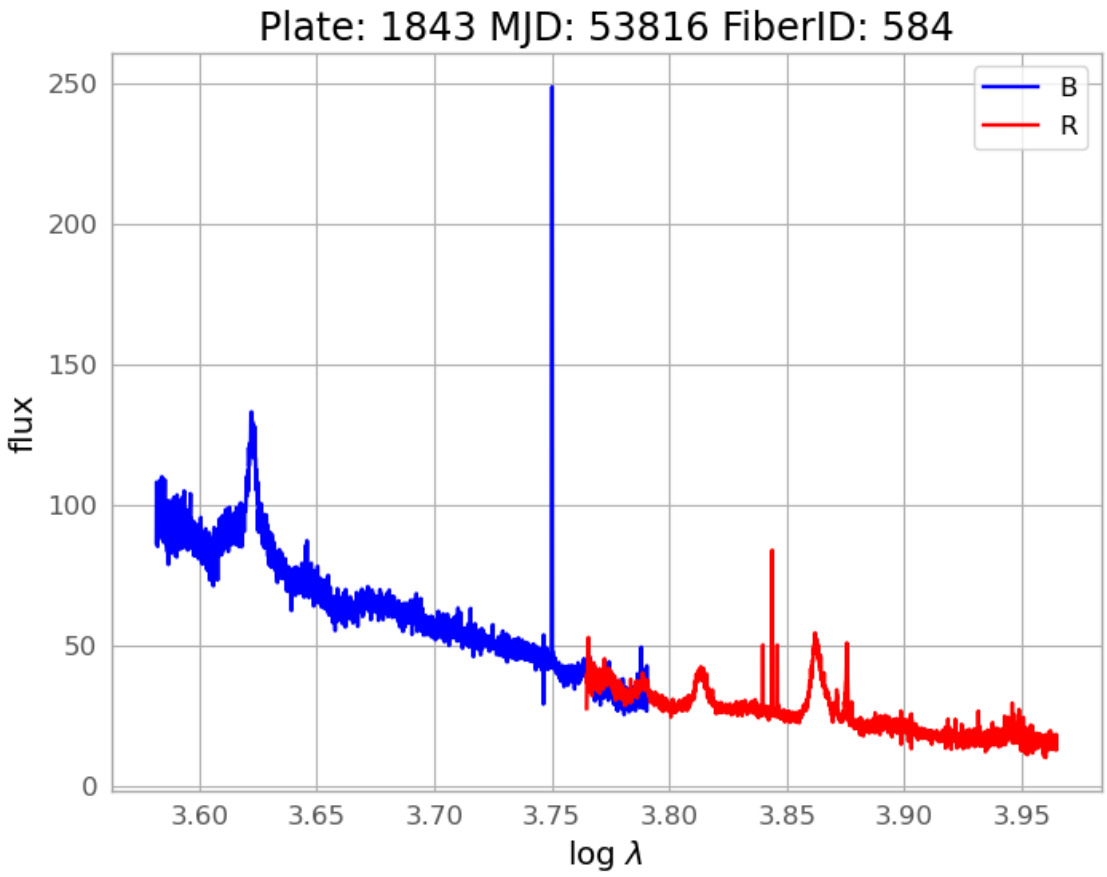


Plate: 1843 MJD: 53816 FiberID: 584
B
R
flux
log λ
250
200
150
100
50
0
3.60
3.65
3.70
3.75
3.80
3.85
3.90
3.95

```
#  11
```

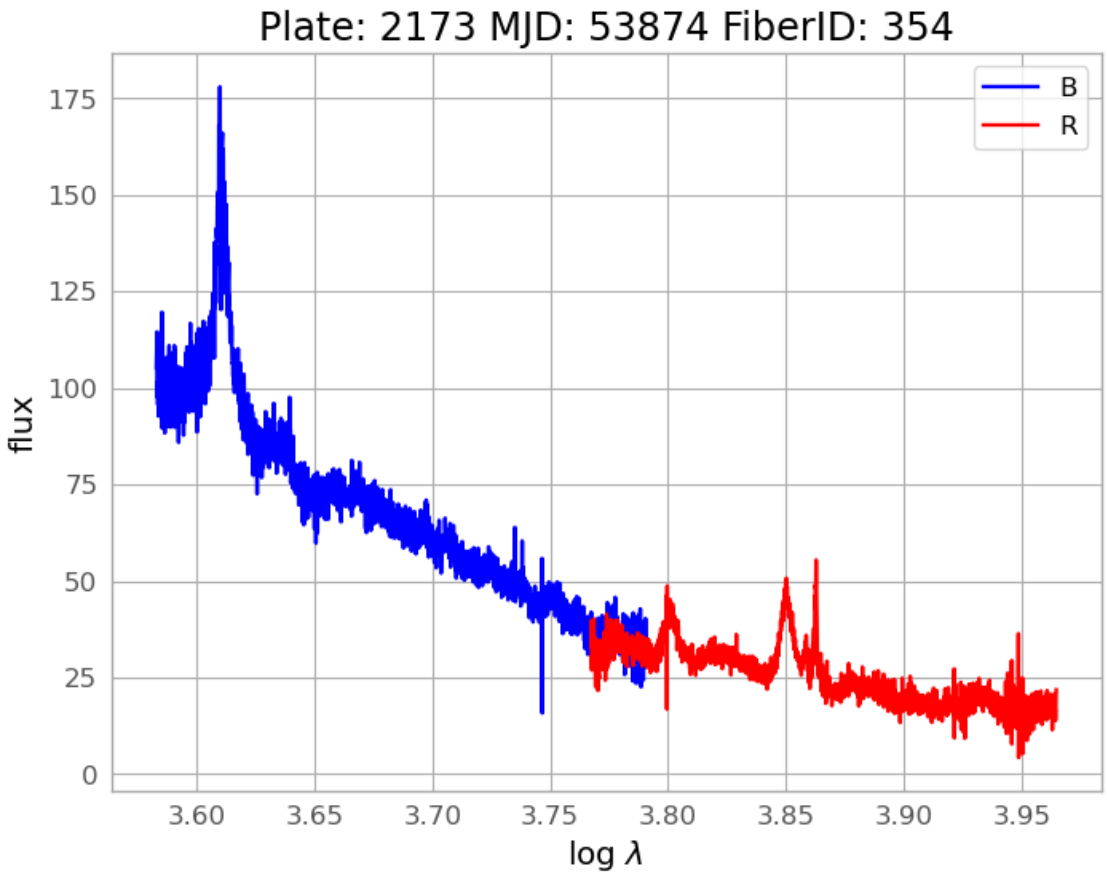


Plate: 2173 MJD: 53874 FiberID: 354
B
R
flux
log λ
175
150
125
100
75
50
25
0
3.60
3.65
3.70
3.75
3.80
3.85
3.90
3.95

```
#  12
```

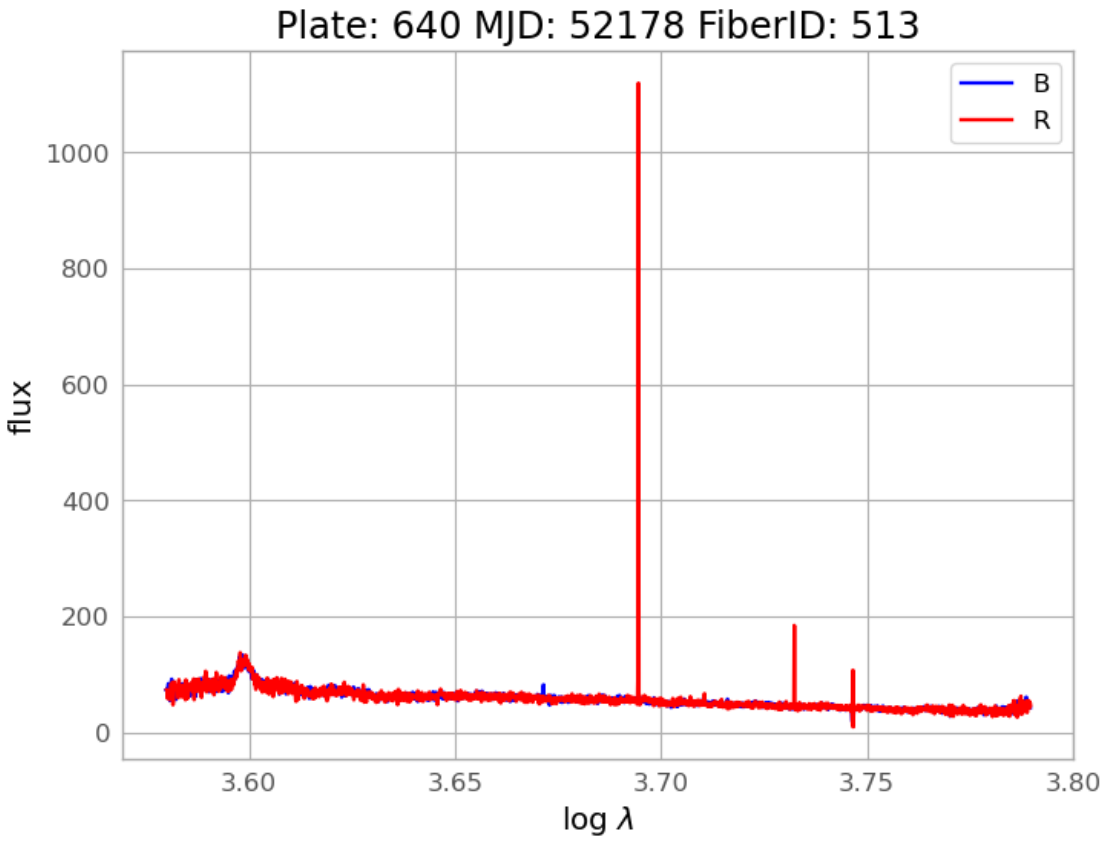


Plate: 640 MJD: 52178 FiberID: 513
B
R
flux
1000
800
600
400
200
0
3.60
3.65
3.70
3.75
3.80
log λ

```
#  13
```

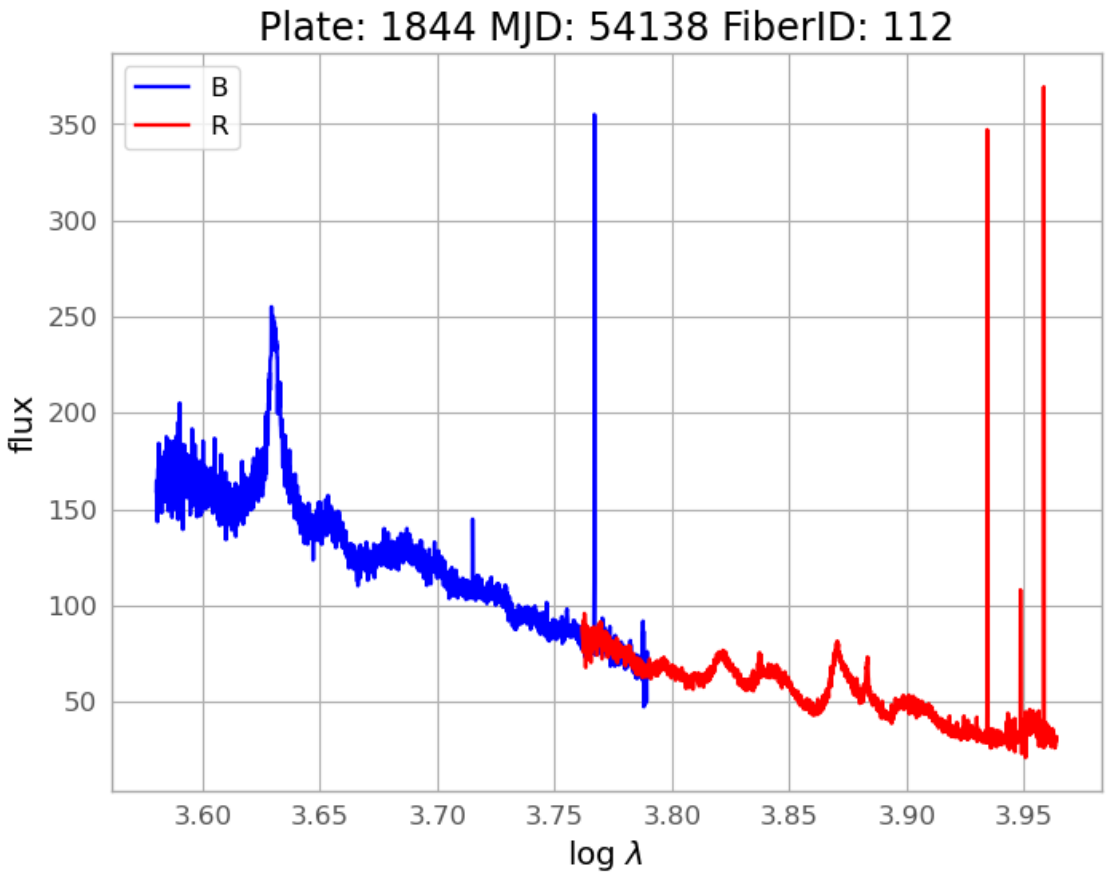


Plate: 1844 MJD: 54138 FiberID: 112
B
R
flux
350
300
250
200
150
100
50
3.60
3.65
3.70
3.75
3.80
3.85
3.90
3.95
log λ

```
#  14
```

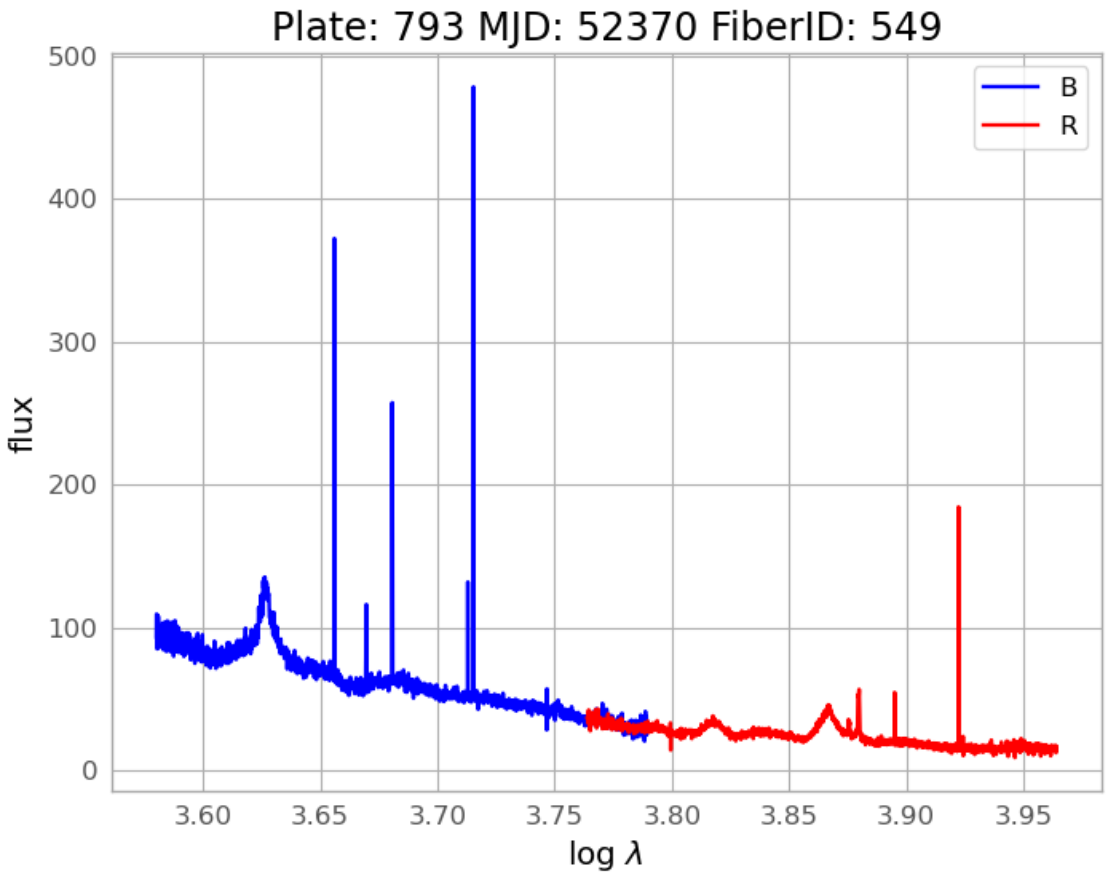


Plate: 793 MJD: 52370 FiberID: 549
B
R
flux
log λ
500
400
300
200
100
0
3.60
3.65
3.70
3.75
3.80
3.85
3.90
3.95

```
#  15
```

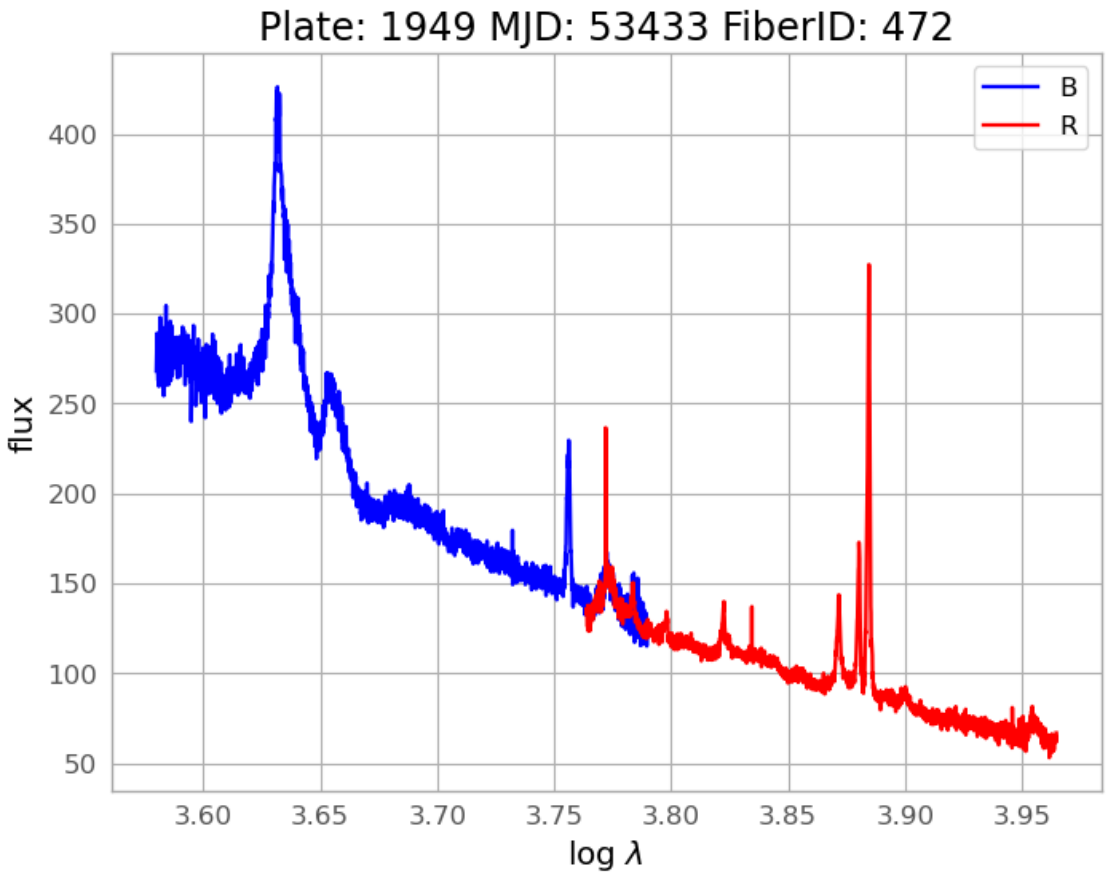


Plate: 1949 MJD: 53433 FiberID: 472
B
R
flux
400
350
300
250
200
150
100
50
3.60
3.65
3.70
3.75
3.80
3.85
3.90
3.95
log λ

```
#  16
```

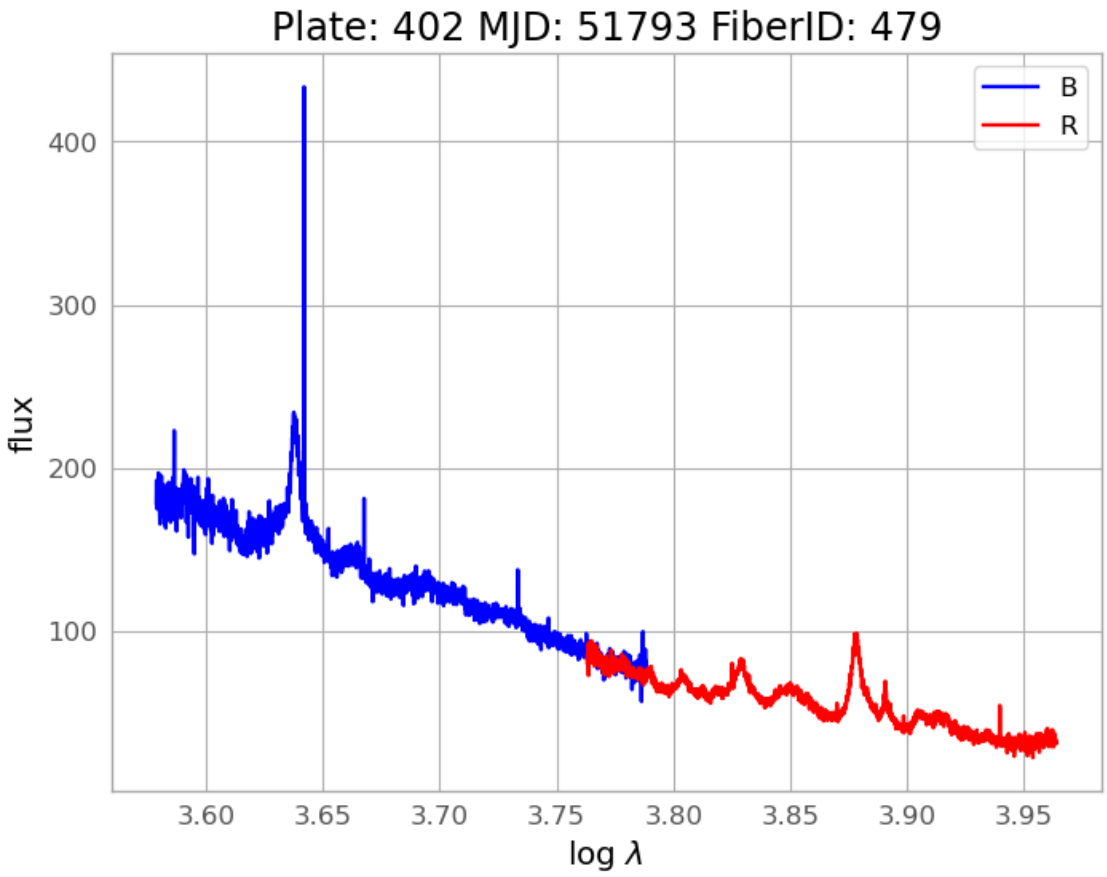


Plate: 402 MJD: 51793 FiberID: 479
B
R
flux
400
300
200
100
3.60
3.65
3.70
3.75
3.80
3.85
3.90
3.95
log λ

```
#  17
```

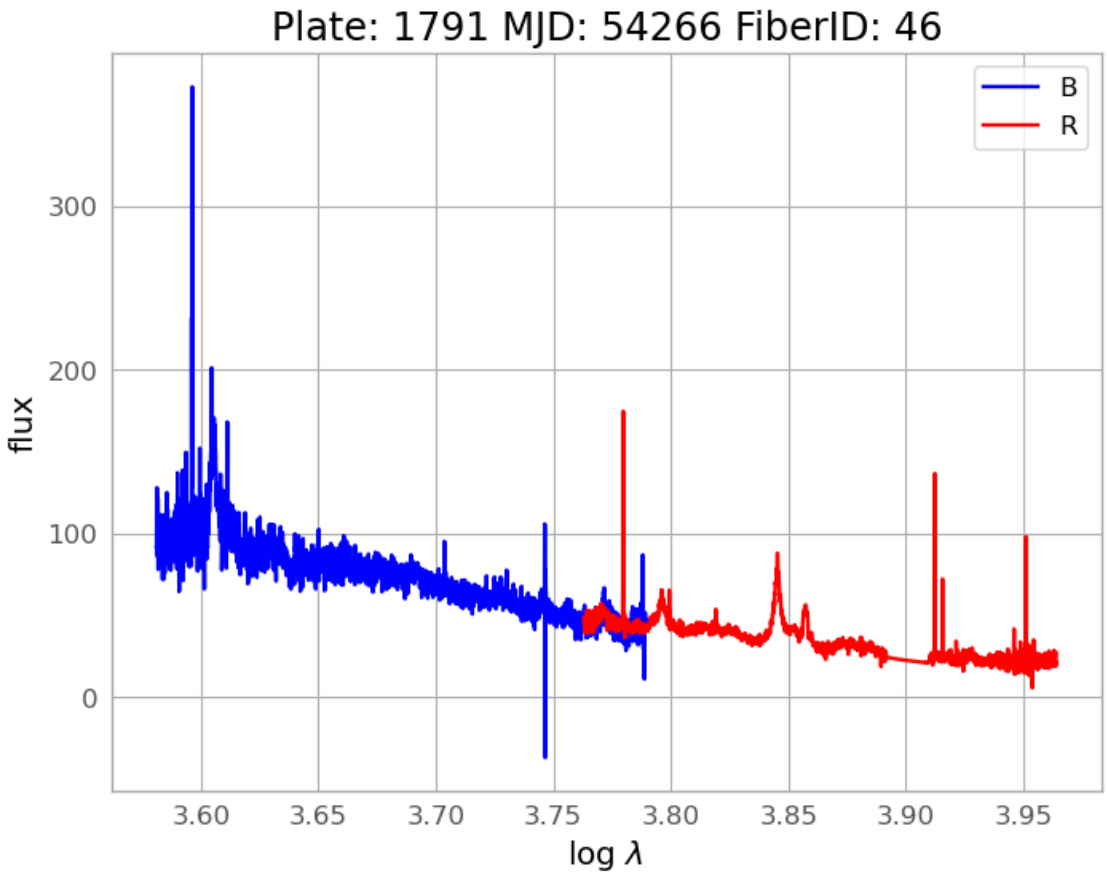


Plate: 1791 MJD: 54266 FiberID: 46
B
R
flux
300
200
100
0
3.60
3.65
3.70
3.75
3.80
3.85
3.90
3.95
log λ

```
#  18
```

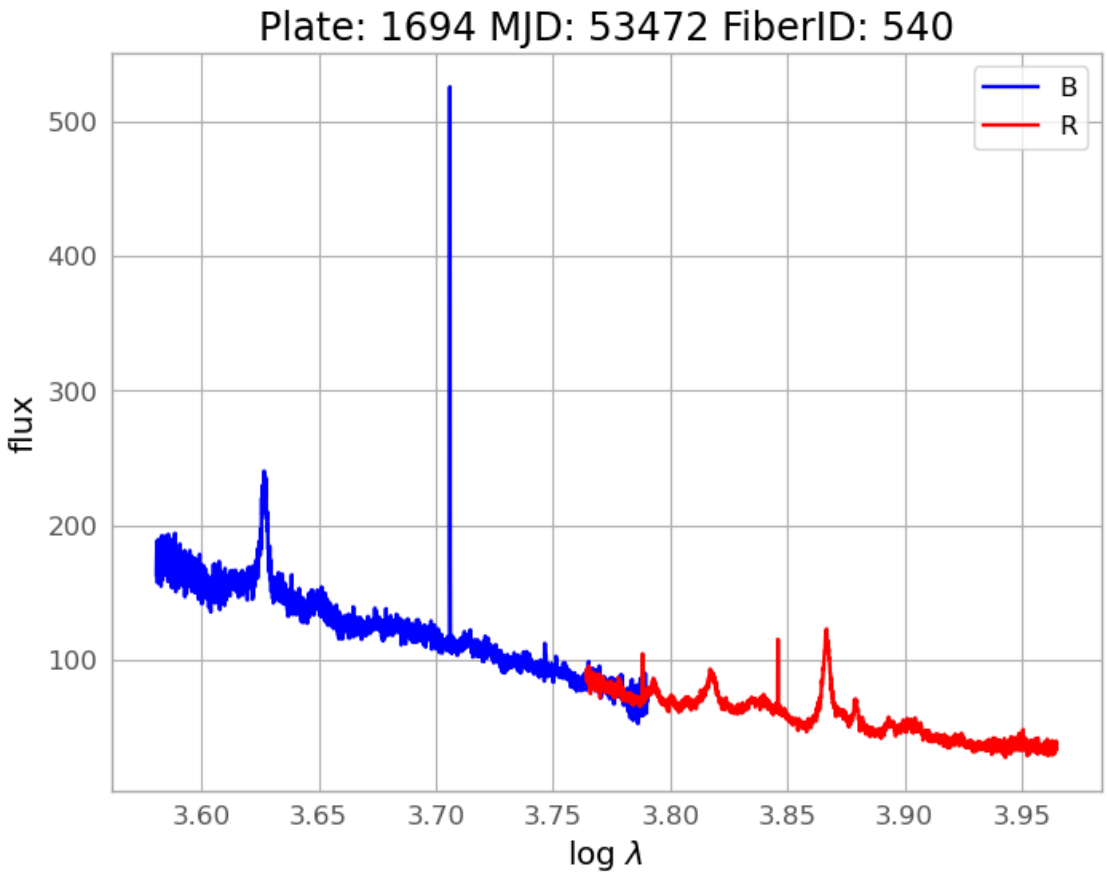


Plate: 1694 MJD: 53472 FiberID: 540
B
R
flux
500
400
300
200
100
3.60
3.65
3.70
3.75
3.80
3.85
3.90
3.95
log λ

```
#  19
```

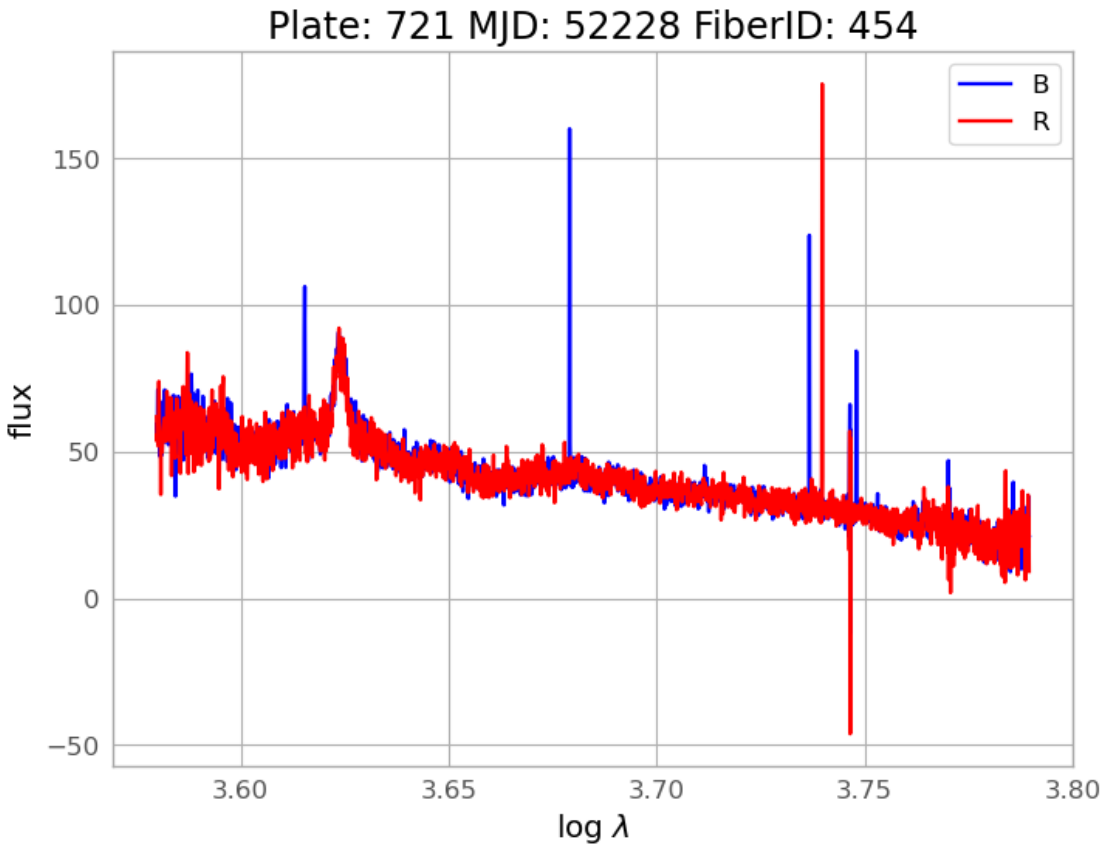


Plate: 721 MJD: 52228 FiberID: 454
B
R
150
100
50
0
−50
flux
3.60
3.65
3.70
3.75
3.80
log λ

20

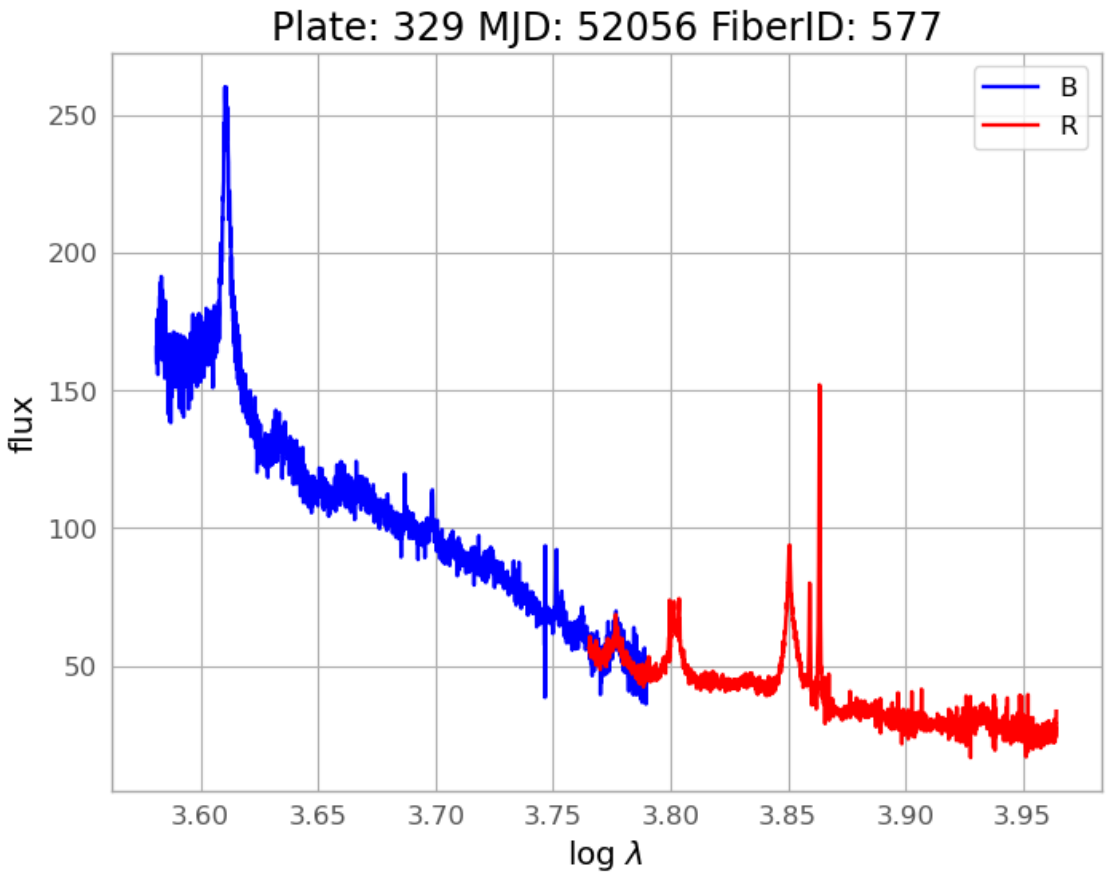


Plate: 329 MJD: 52056 FiberID: 577
B
R
flux
250
200
150
100
50
3.60
3.65
3.70
3.75
3.80
3.85
3.90
3.95
log λ

```
#  21
```

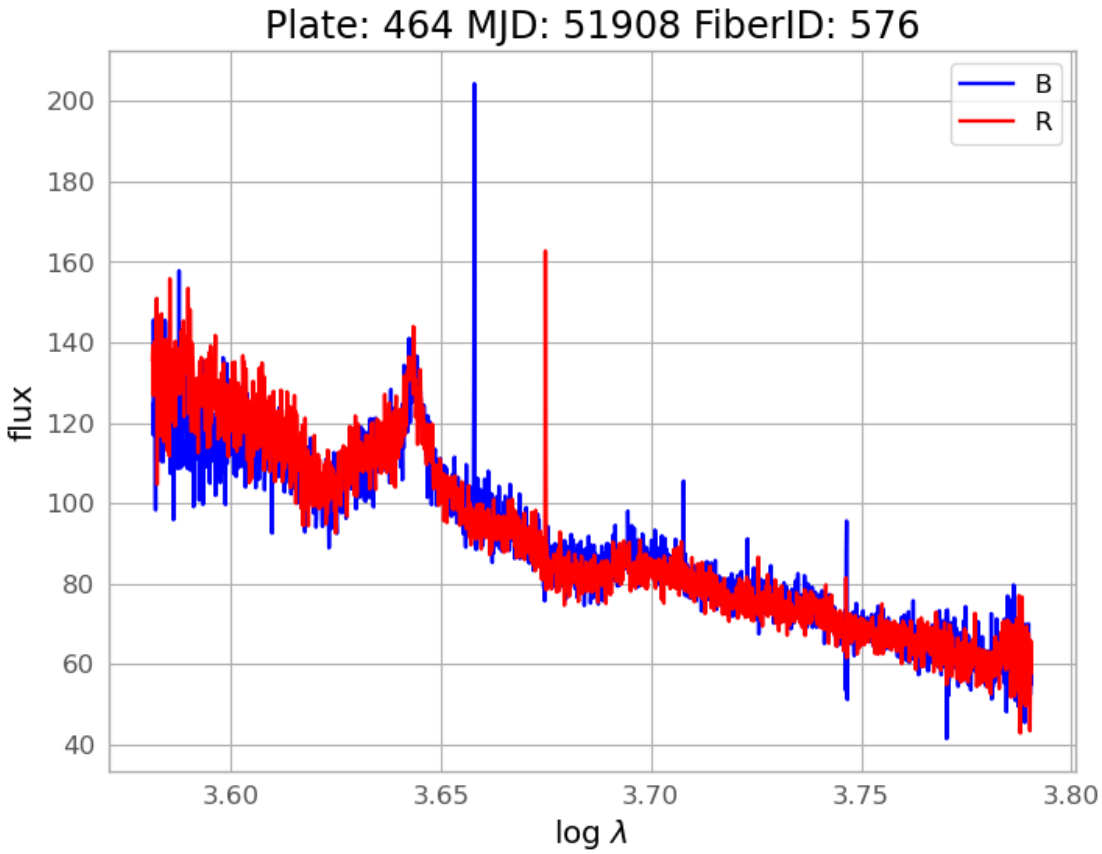


Plate: 464 MJD: 51908 FiberID: 576
B
R
flux
200
180
160
140
120
100
80
60
40
3.60
3.65
3.70
3.75
3.80
log λ

22

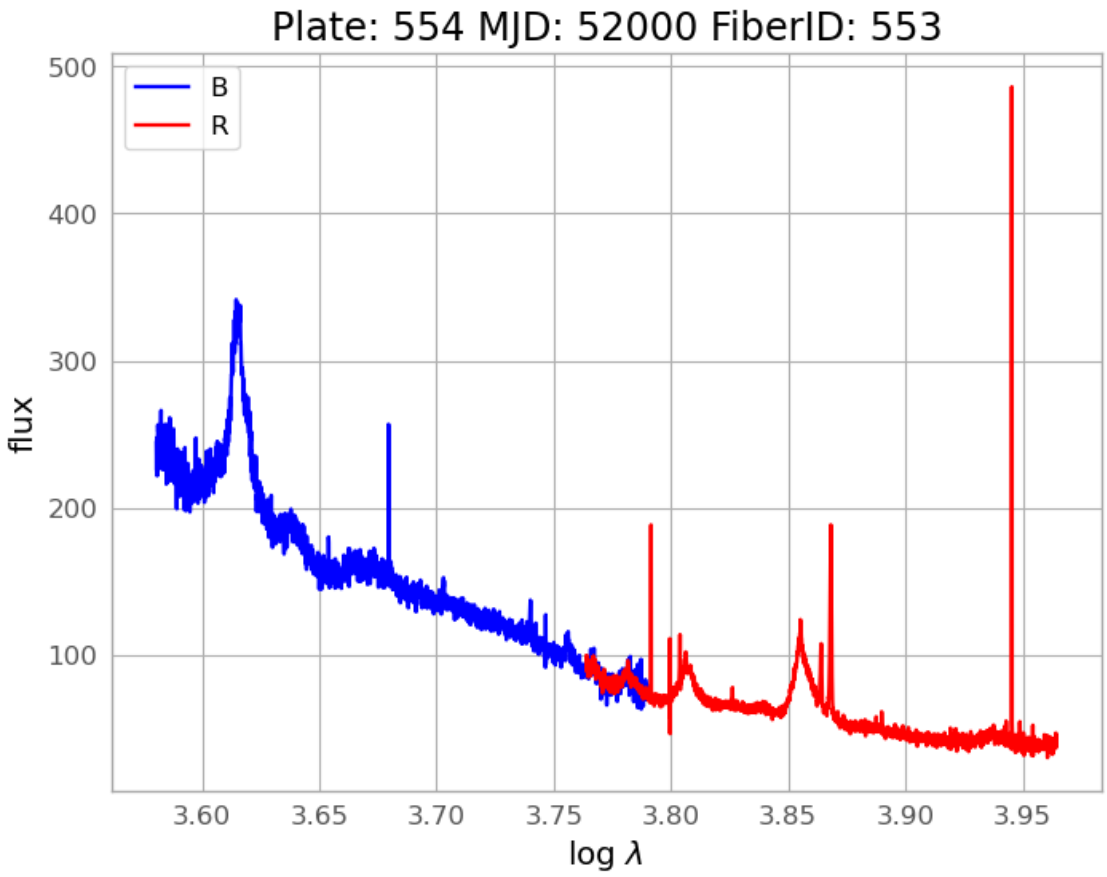


Plate: 554 MJD: 52000 FiberID: 553
B
R
flux
log λ
500
400
300
200
100
3.60
3.65
3.70
3.75
3.80
3.85
3.90
3.95

23

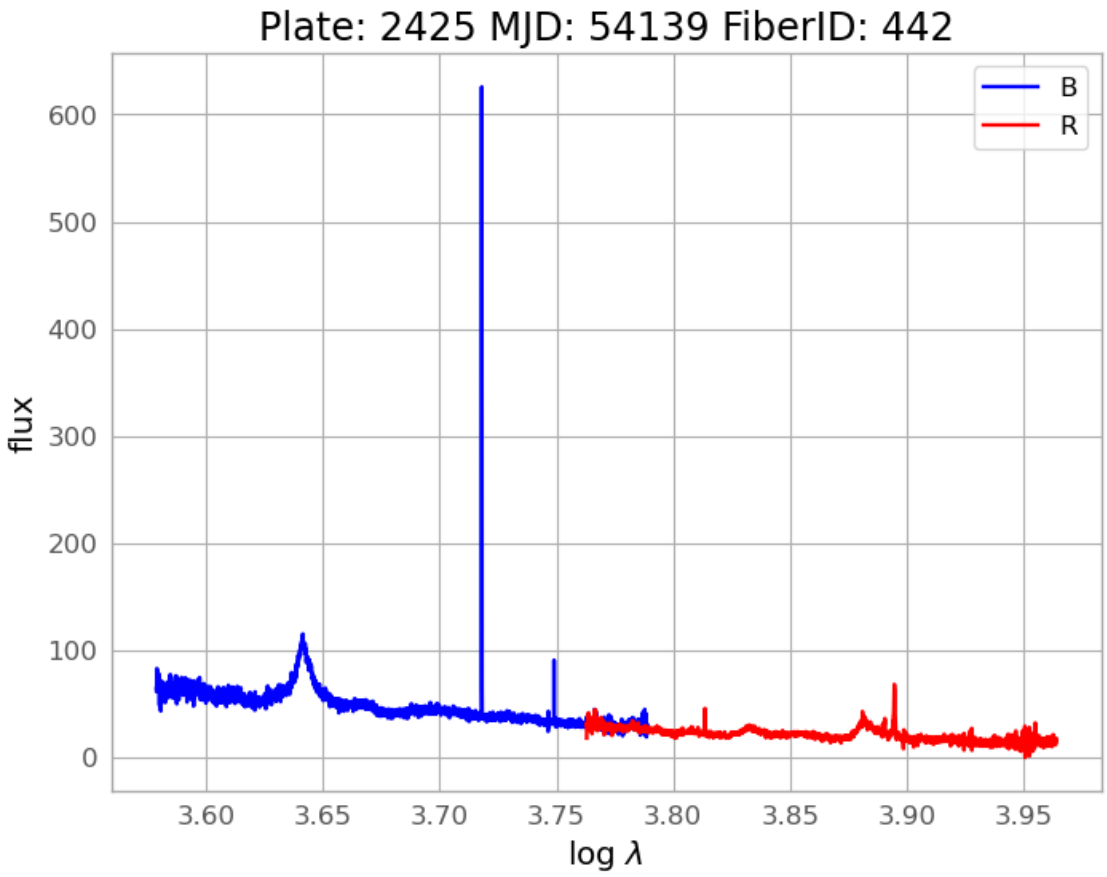


Plate: 2425 MJD: 54139 FiberID: 442
B
R
flux
600
500
400
300
200
100
0
3.60
3.65
3.70
3.75
3.80
3.85
3.90
3.95
log λ

24

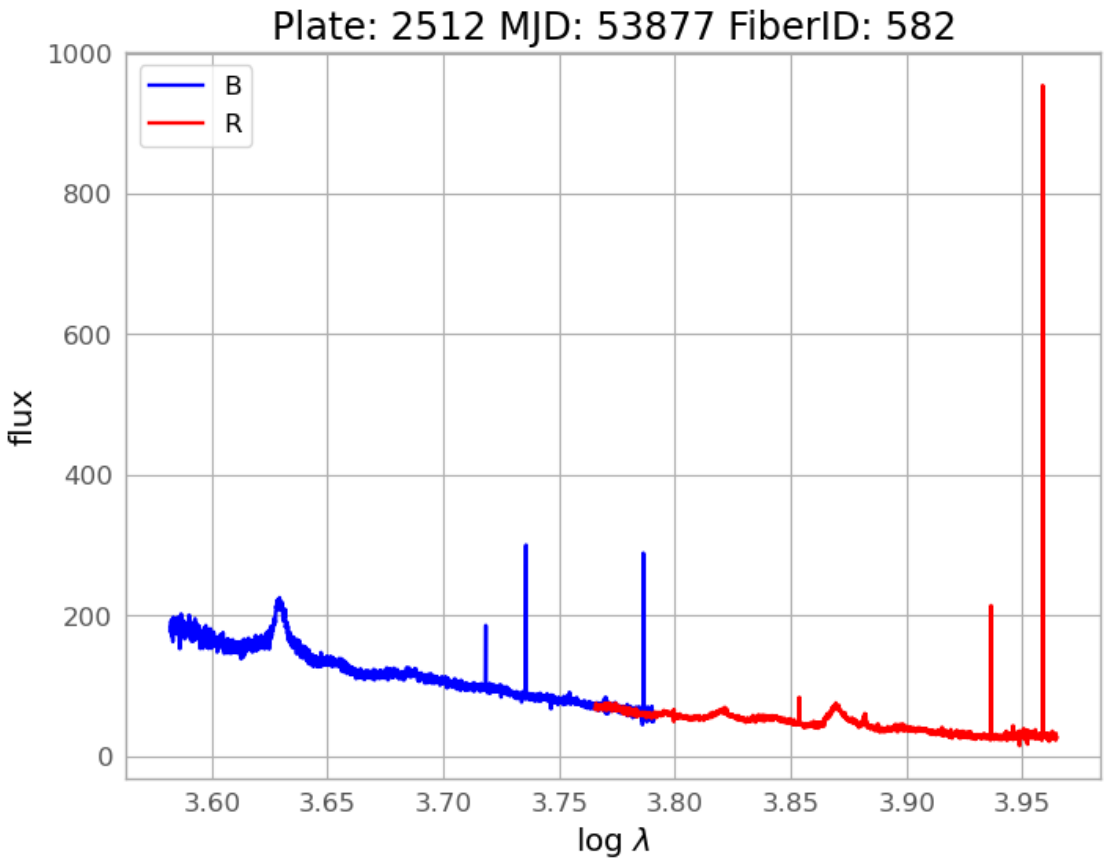


Plate: 2512 MJD: 53877 FiberID: 582
B
R
flux
log λ
1000
800
600
400
200
0
3.60
3.65
3.70
3.75
3.80
3.85
3.90
3.95

25

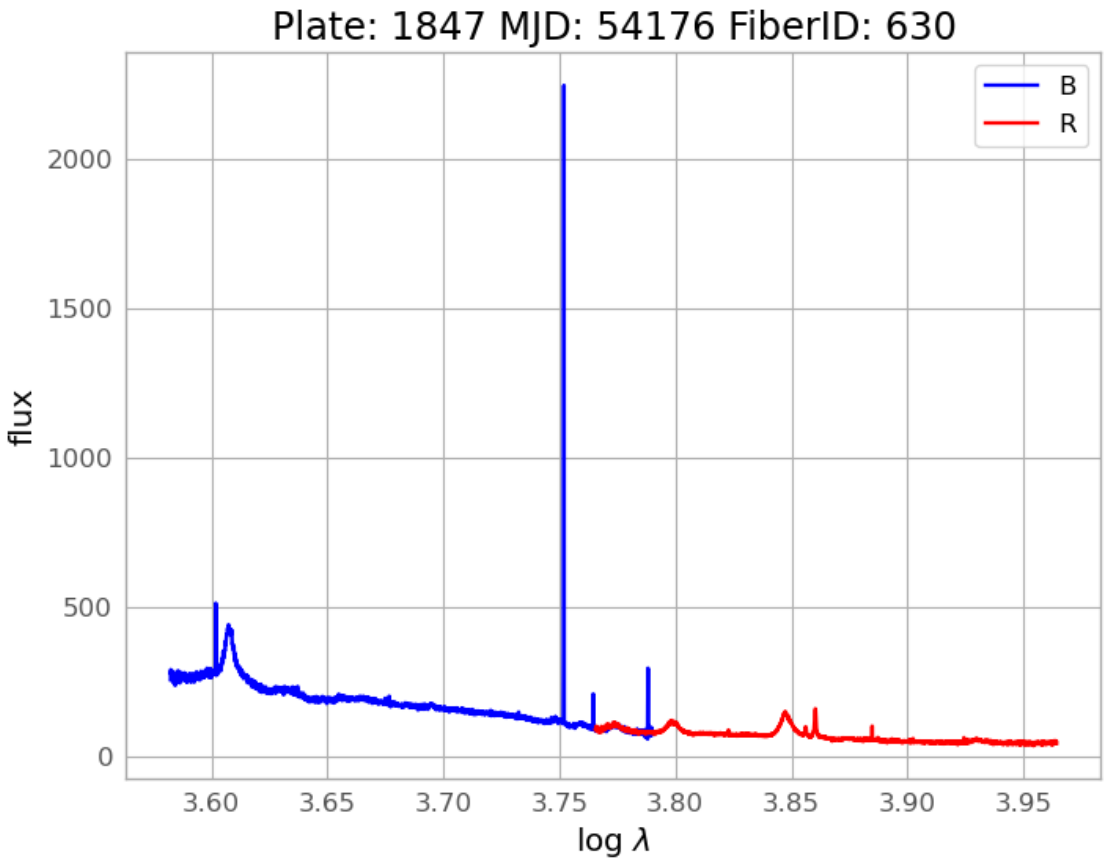


Plate: 1847 MJD: 54176 FiberID: 630
B
R
flux
2000
1500
1000
500
0
3.60
3.65
3.70
3.75
3.80
3.85
3.90
3.95
log λ

26

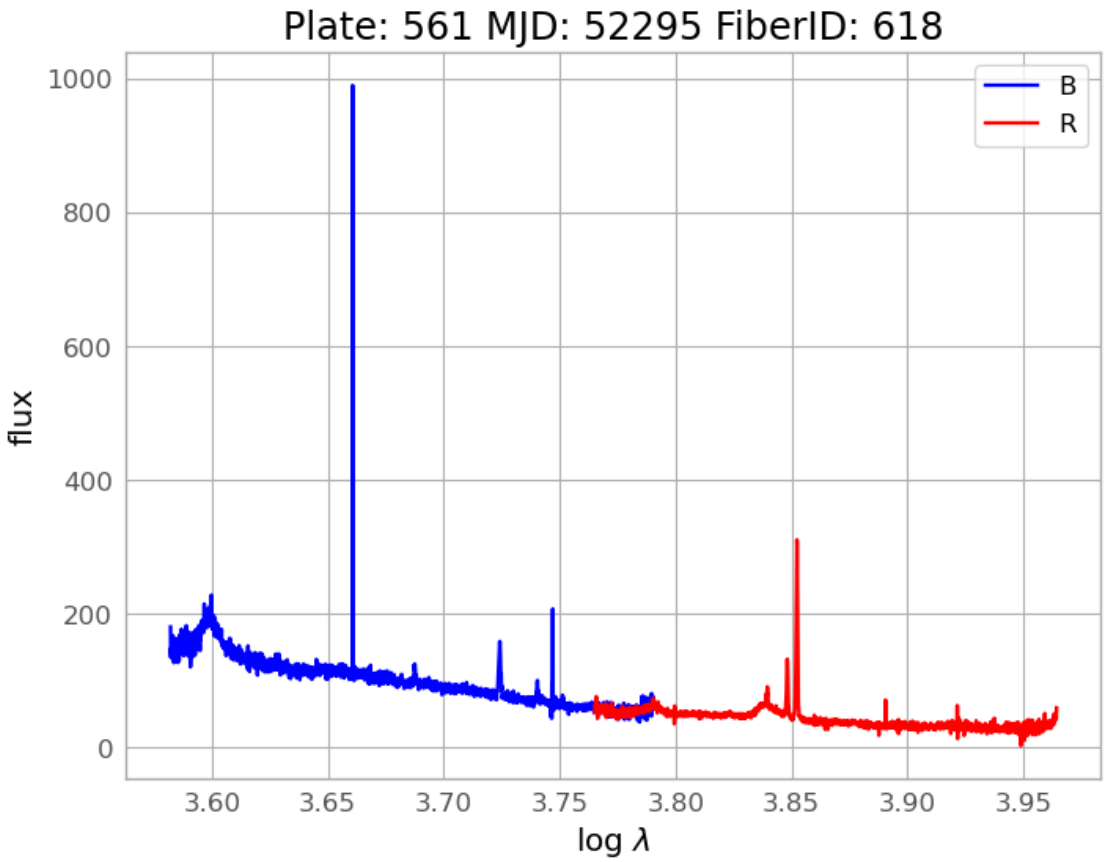


Plate: 561 MJD: 52295 FiberID: 618
B
R
flux
log λ
1000
800
600
400
200
0
3.60
3.65
3.70
3.75
3.80
3.85
3.90
3.95

27

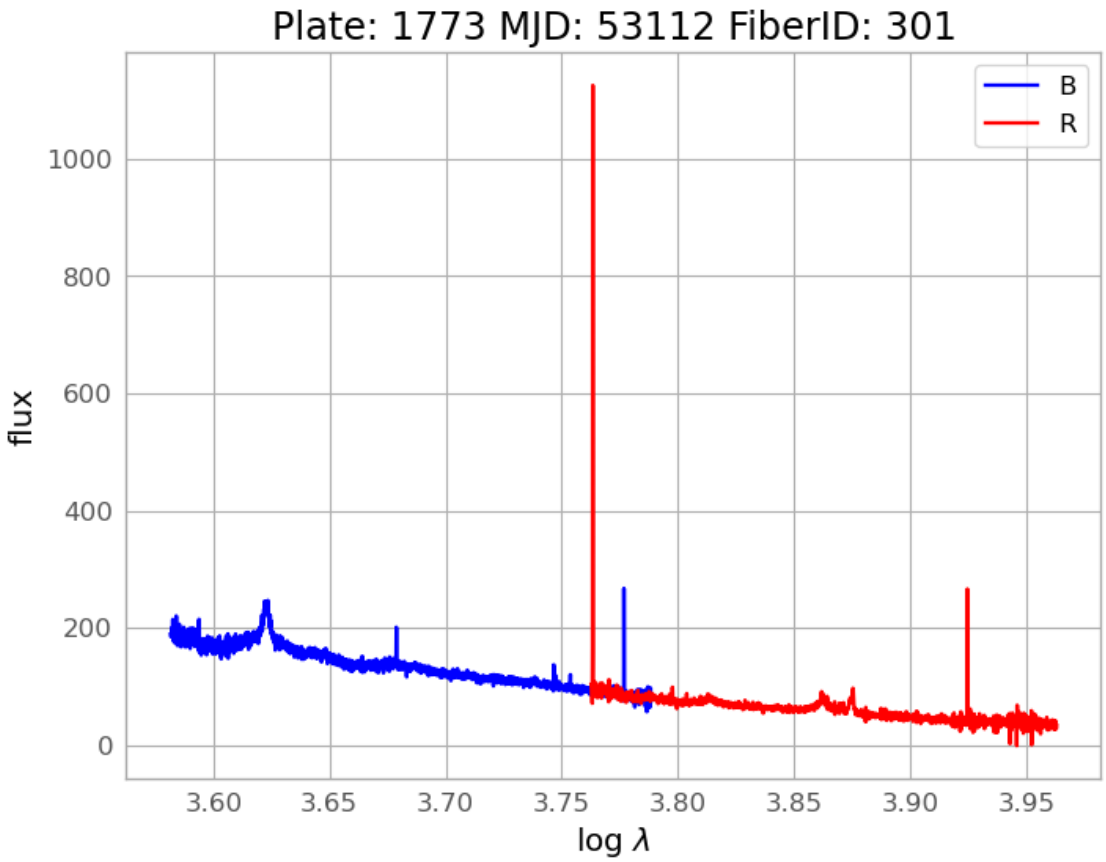


Plate: 1773 MJD: 53112 FiberID: 301
B
R
flux
1000
800
600
400
200
0
3.60
3.65
3.70
3.75
3.80
3.85
3.90
3.95
log λ

28

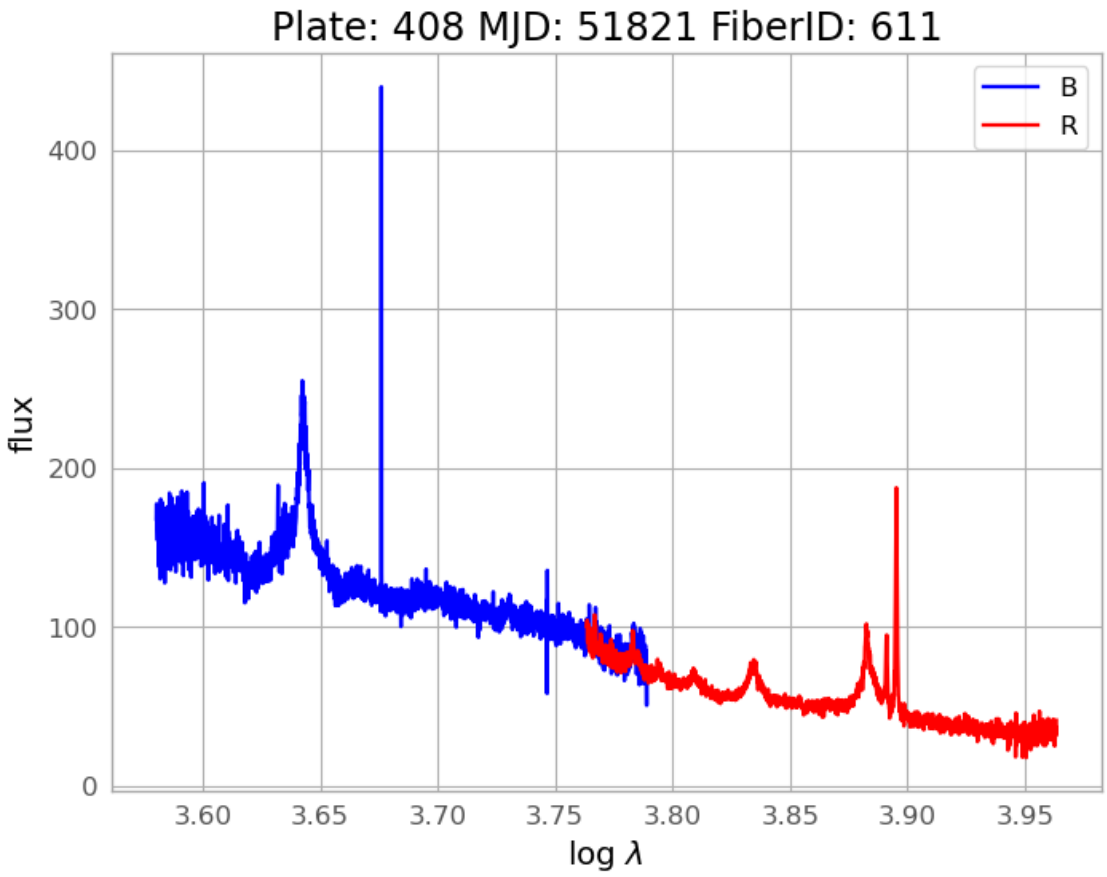


Plate: 408 MJD: 51821 FiberID: 611
B
R
flux
400
300
200
100
0
3.60
3.65
3.70
3.75
3.80
3.85
3.90
3.95
log λ

29

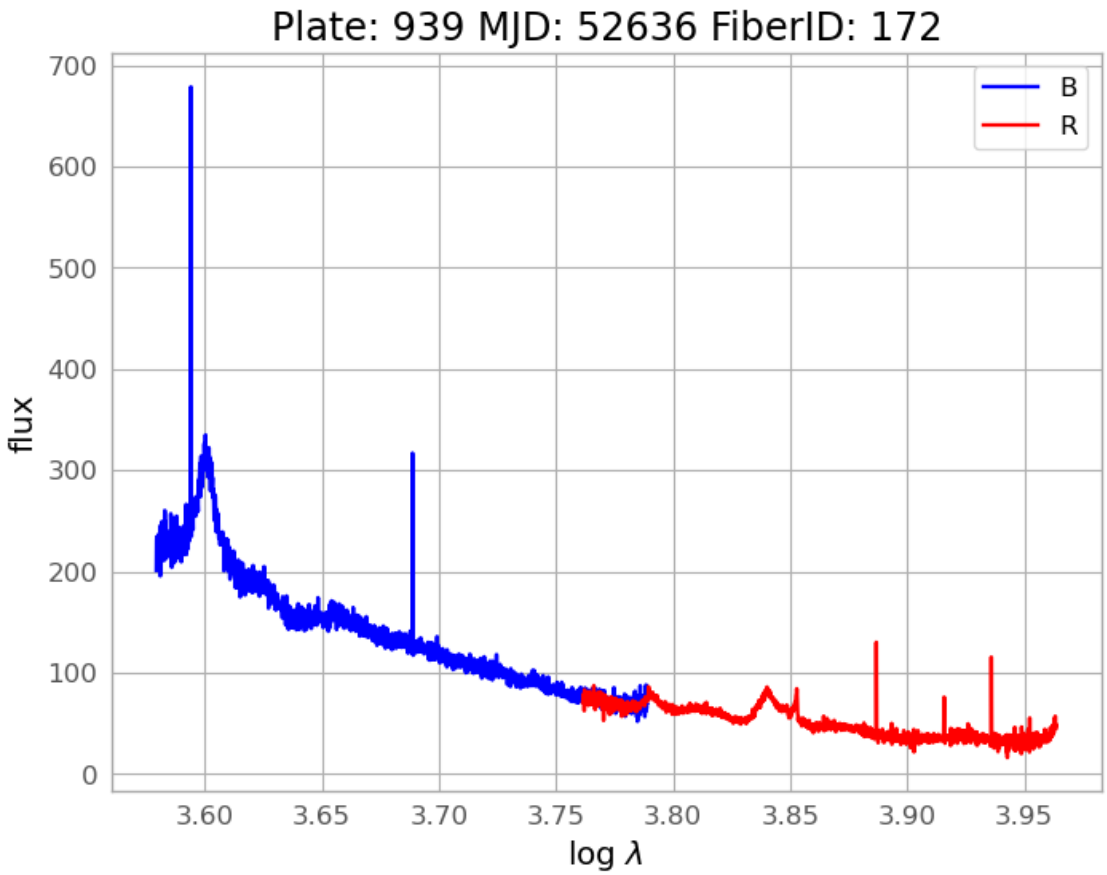


Plate: 939 MJD: 52636 FiberID: 172
B
R
flux
log λ
700
600
500
400
300
200
100
0
3.60
3.65
3.70
3.75
3.80
3.85
3.90
3.95

30

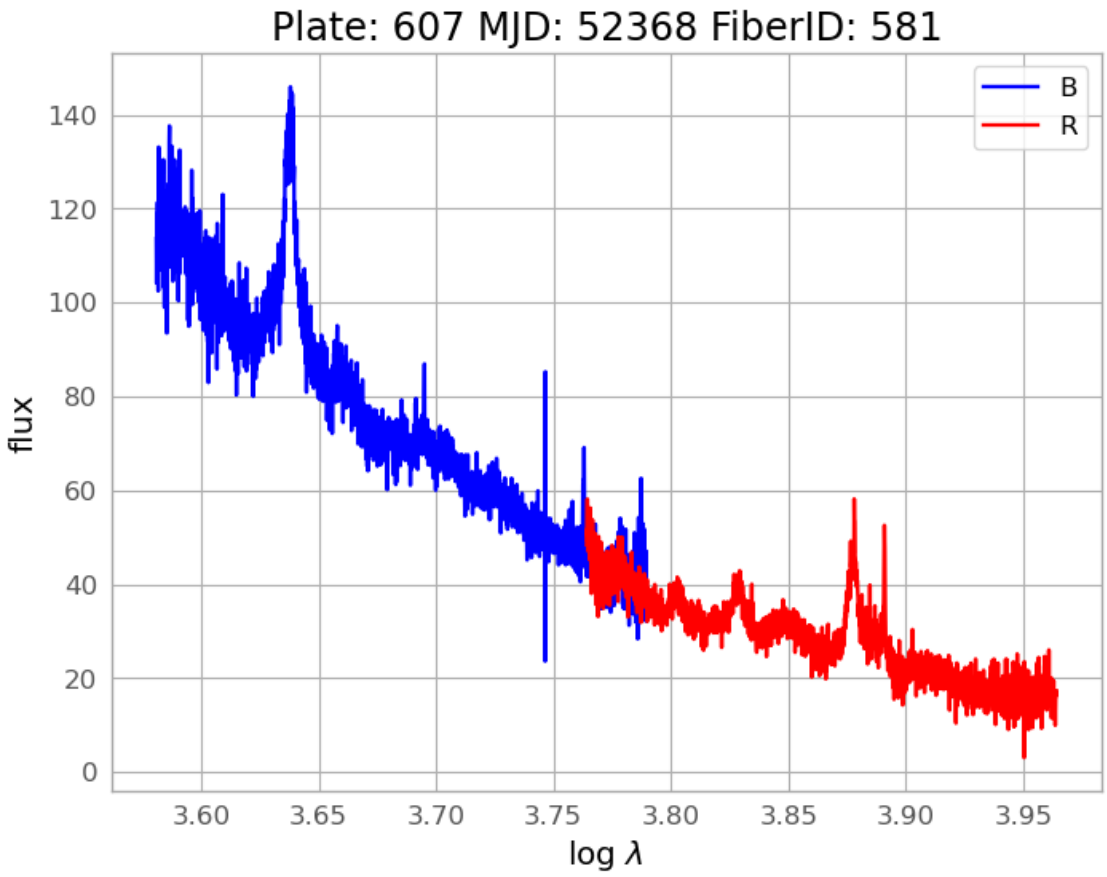


Plate: 607 MJD: 52368 FiberID: 581
B
R
flux
log λ
0
20
40
60
80
100
120
140
3.60
3.65
3.70
3.75
3.80
3.85
3.90
3.95

```
#  31
```

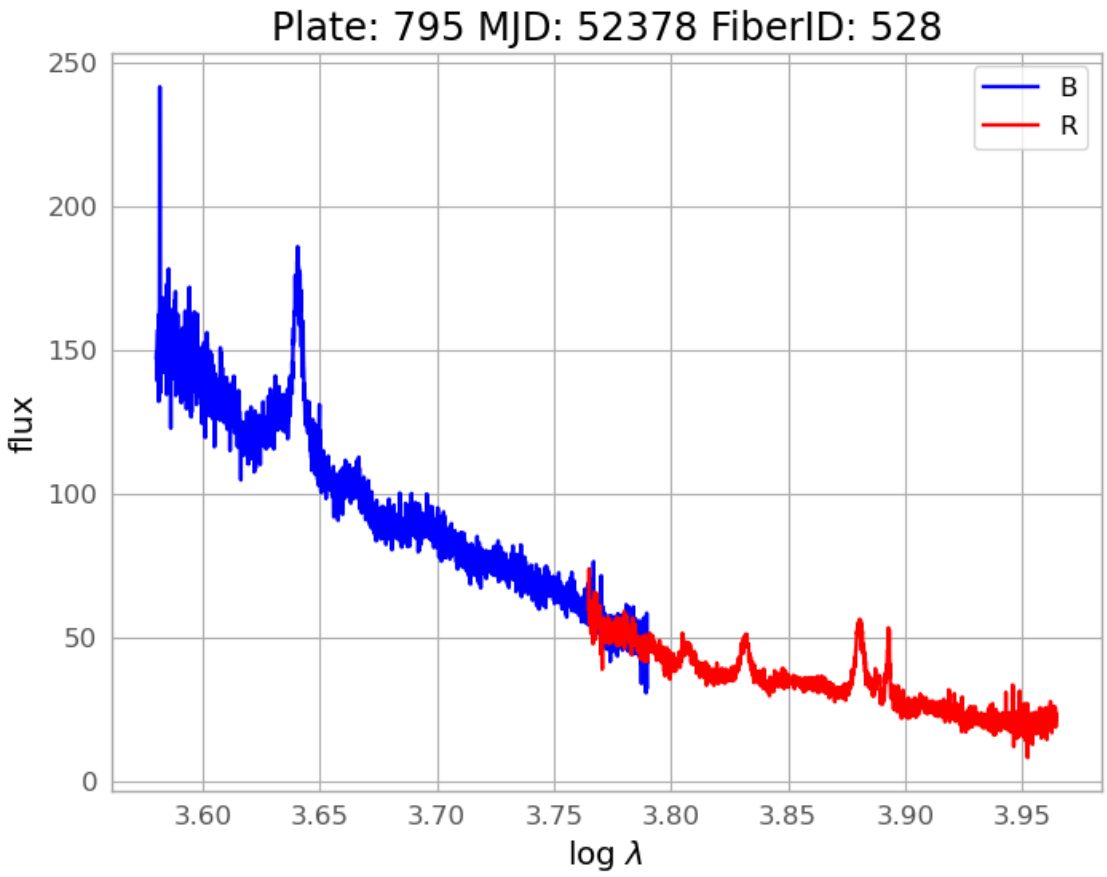


Plate: 795 MJD: 52378 FiberID: 528
B
R
flux
log λ
250
200
150
100
50
0
3.60
3.65
3.70
3.75
3.80
3.85
3.90
3.95

32

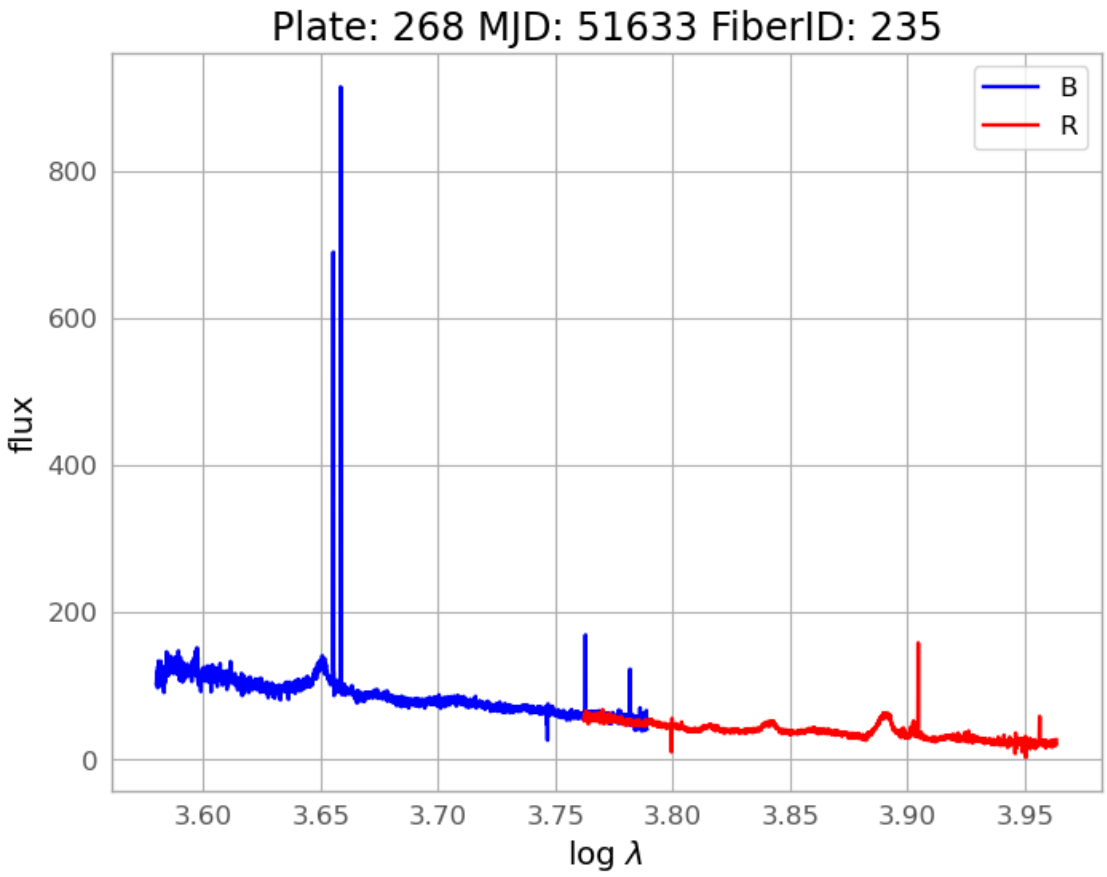


Plate: 268 MJD: 51633 FiberID: 235
B
R
flux
800
600
400
200
0
3.60
3.65
3.70
3.75
3.80
3.85
3.90
3.95
log λ

33

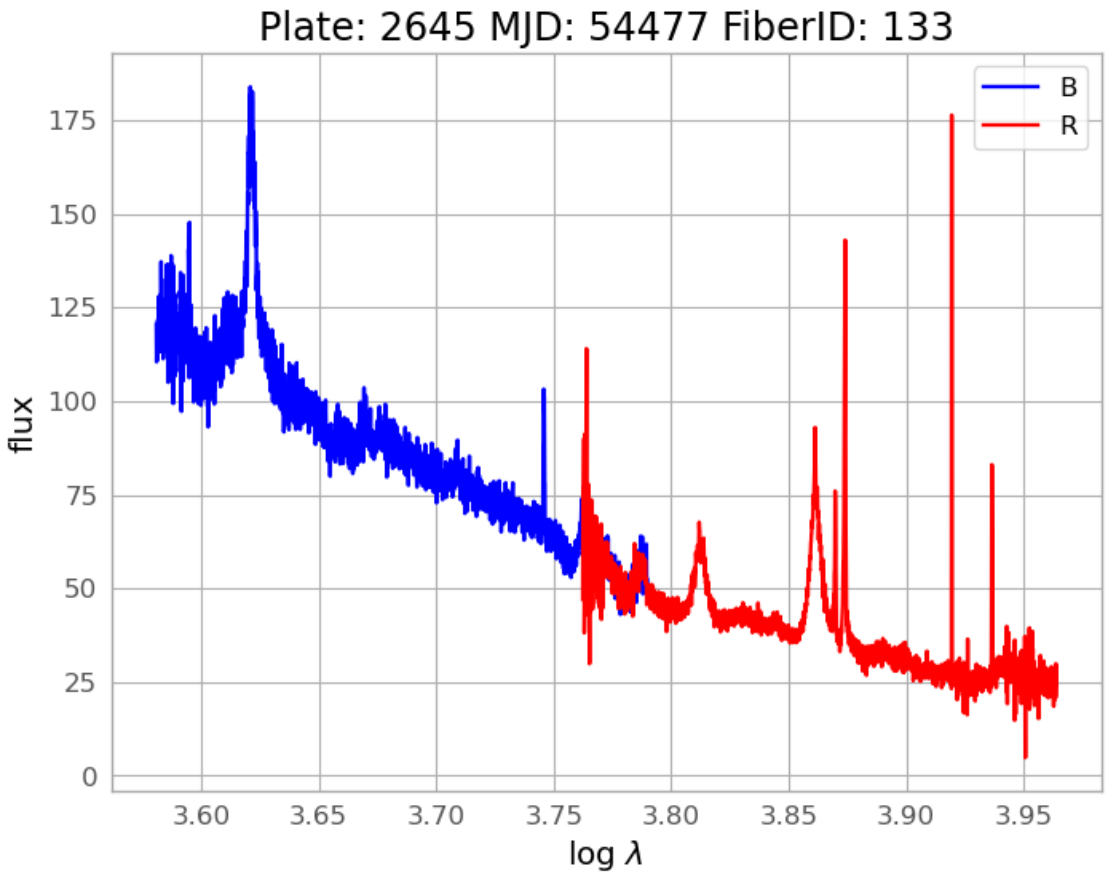


Plate: 2645 MJD: 54477 FiberID: 133
B
R
flux
log λ
0
25
50
75
100
125
150
175
3.60
3.65
3.70
3.75
3.80
3.85
3.90
3.95

34

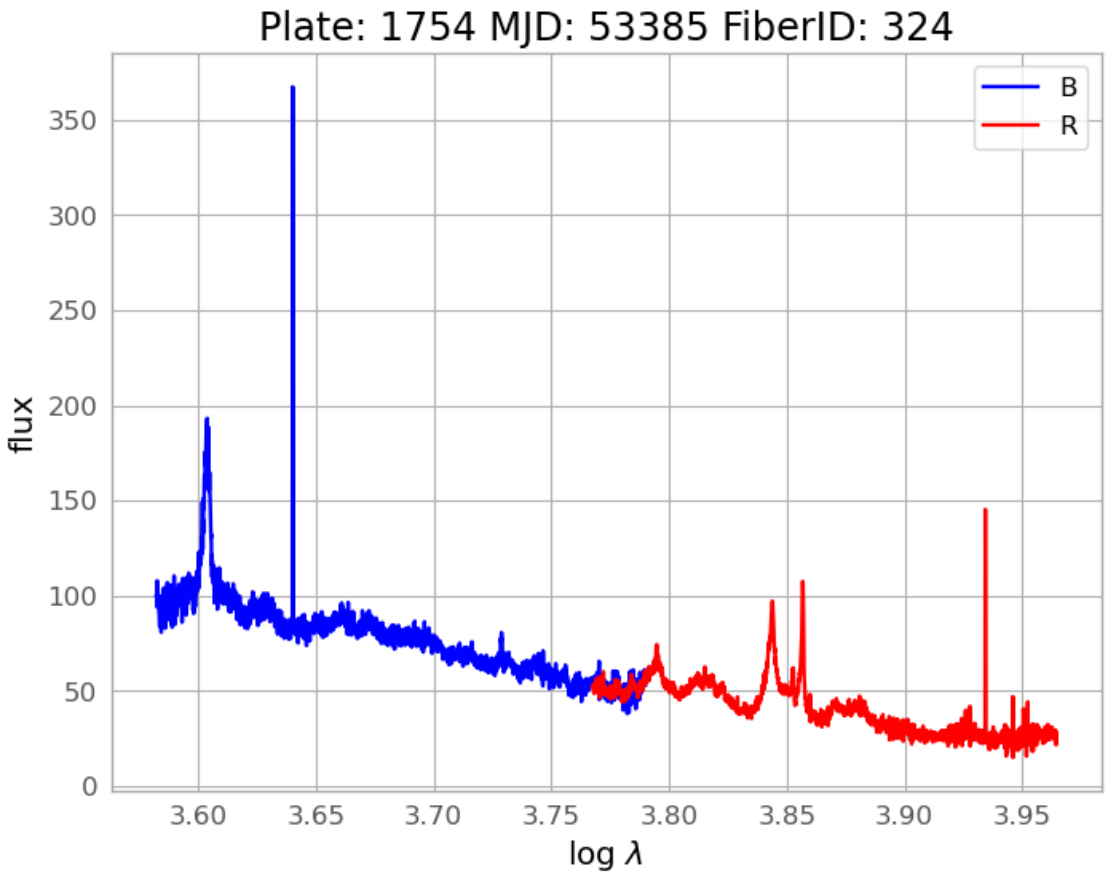


Plate: 1754 MJD: 53385 FiberID: 324
B
R
flux
log λ
0
50
100
150
200
250
300
350
3.60
3.65
3.70
3.75
3.80
3.85
3.90
3.95

35

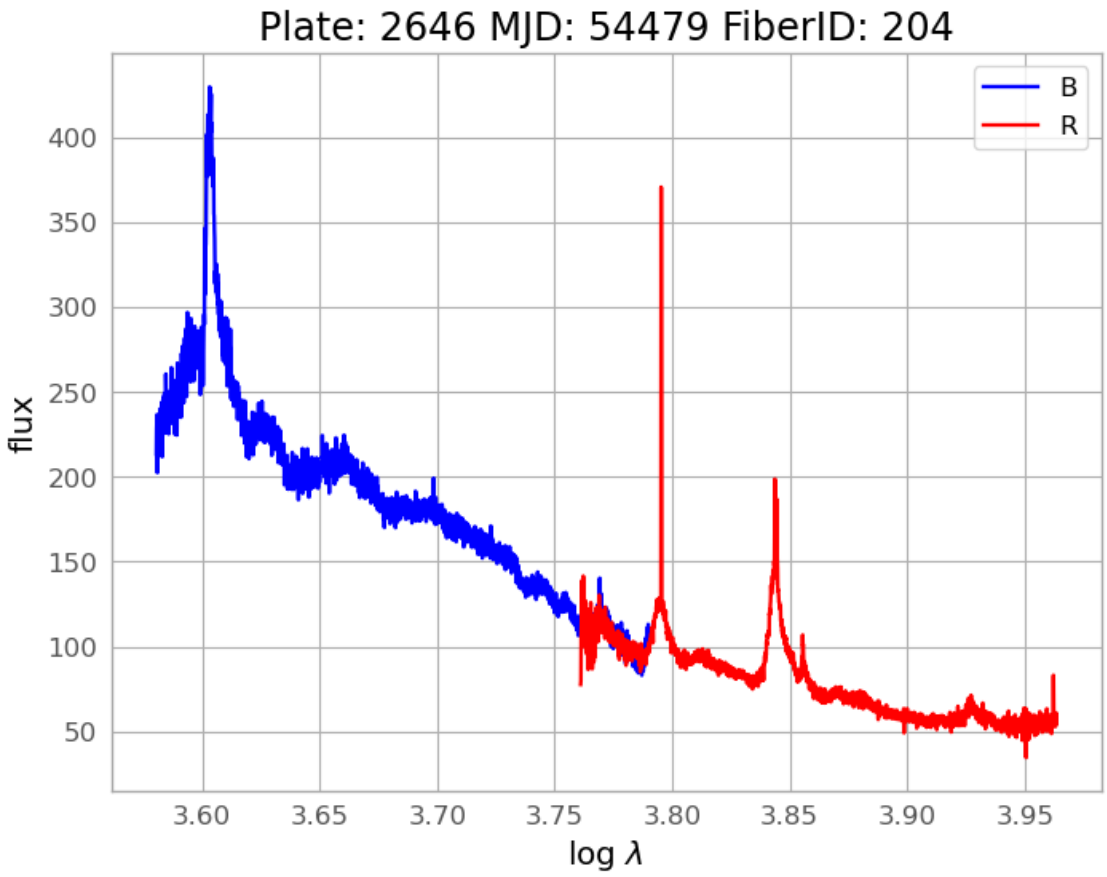


Plate: 2646 MJD: 54479 FiberID: 204
B
R
flux
400
350
300
250
200
150
100
50
3.60
3.65
3.70
3.75
3.80
3.85
3.90
3.95
log λ

36

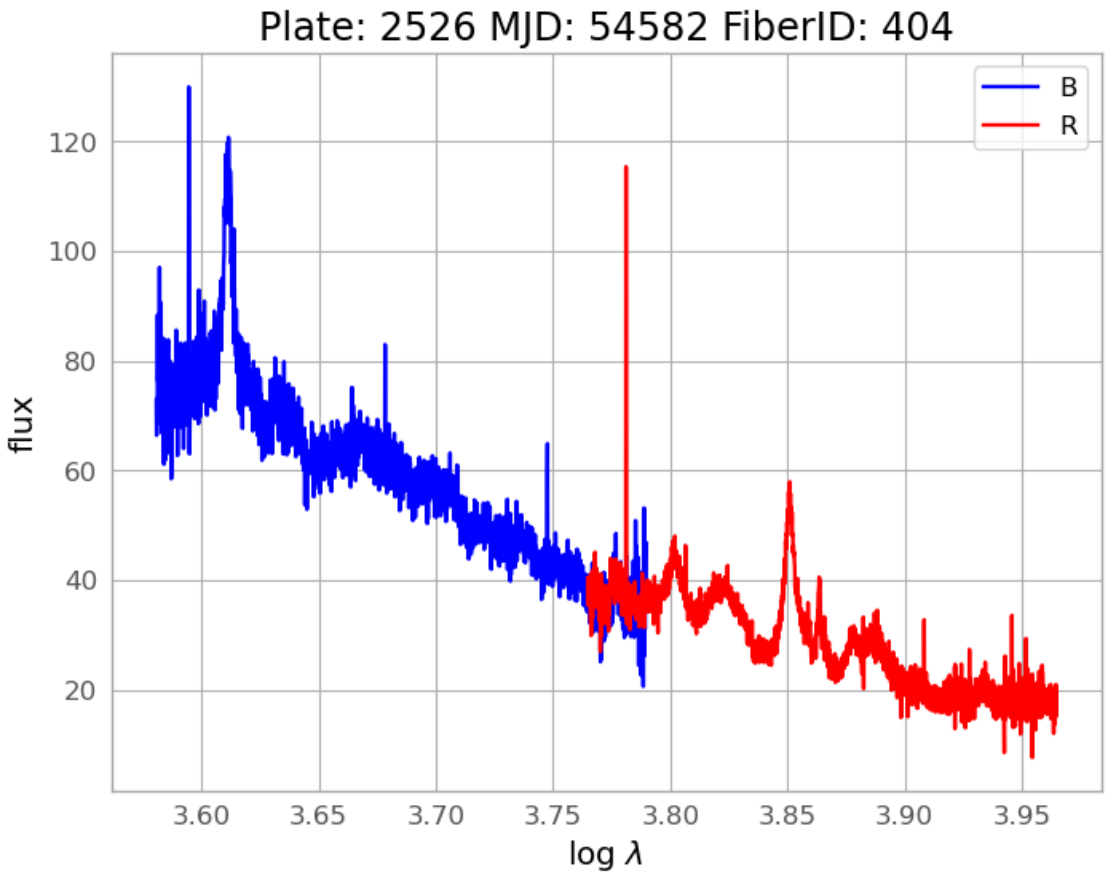


Plate: 2526 MJD: 54582 FiberID: 404
B
R
flux
120
100
80
60
40
20
3.60
3.65
3.70
3.75
3.80
3.85
3.90
3.95
log λ

37

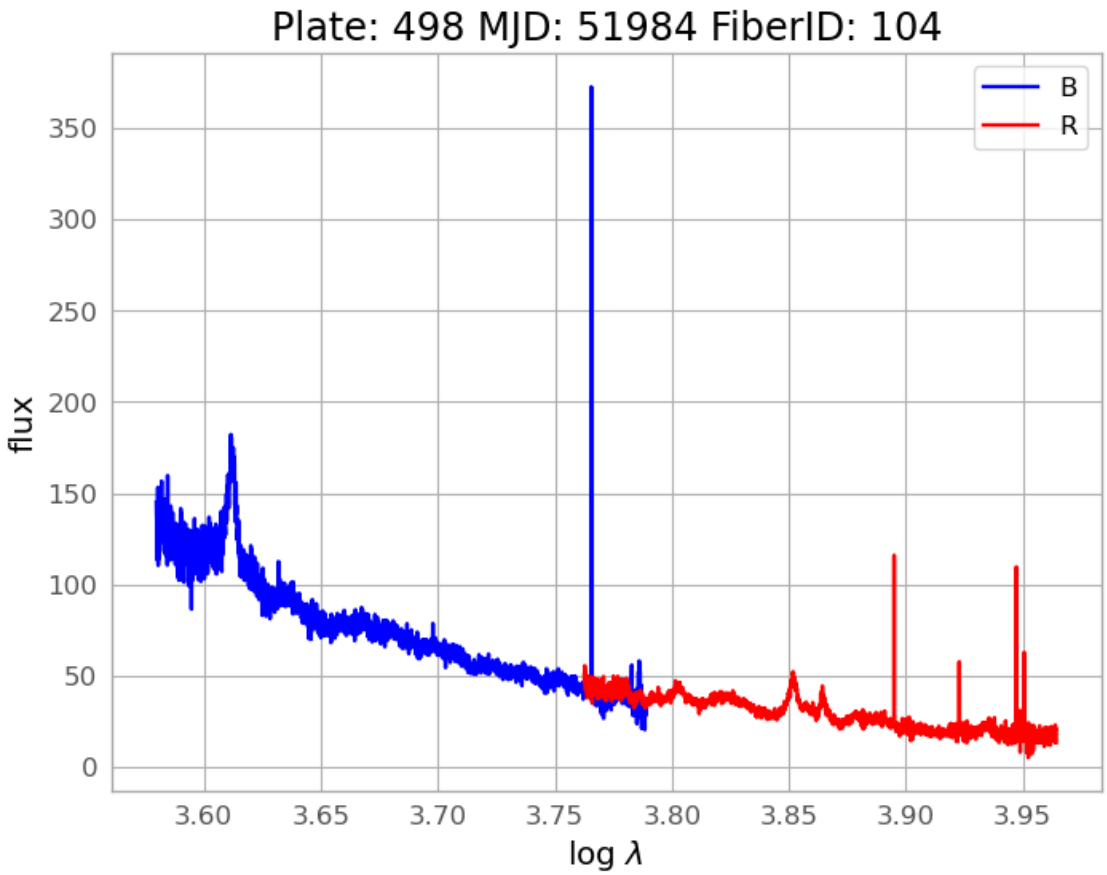


Plate: 498 MJD: 51984 FiberID: 104
B
R
flux
0
50
100
150
200
250
300
350
3.60
3.65
3.70
3.75
3.80
3.85
3.90
3.95
log λ

38

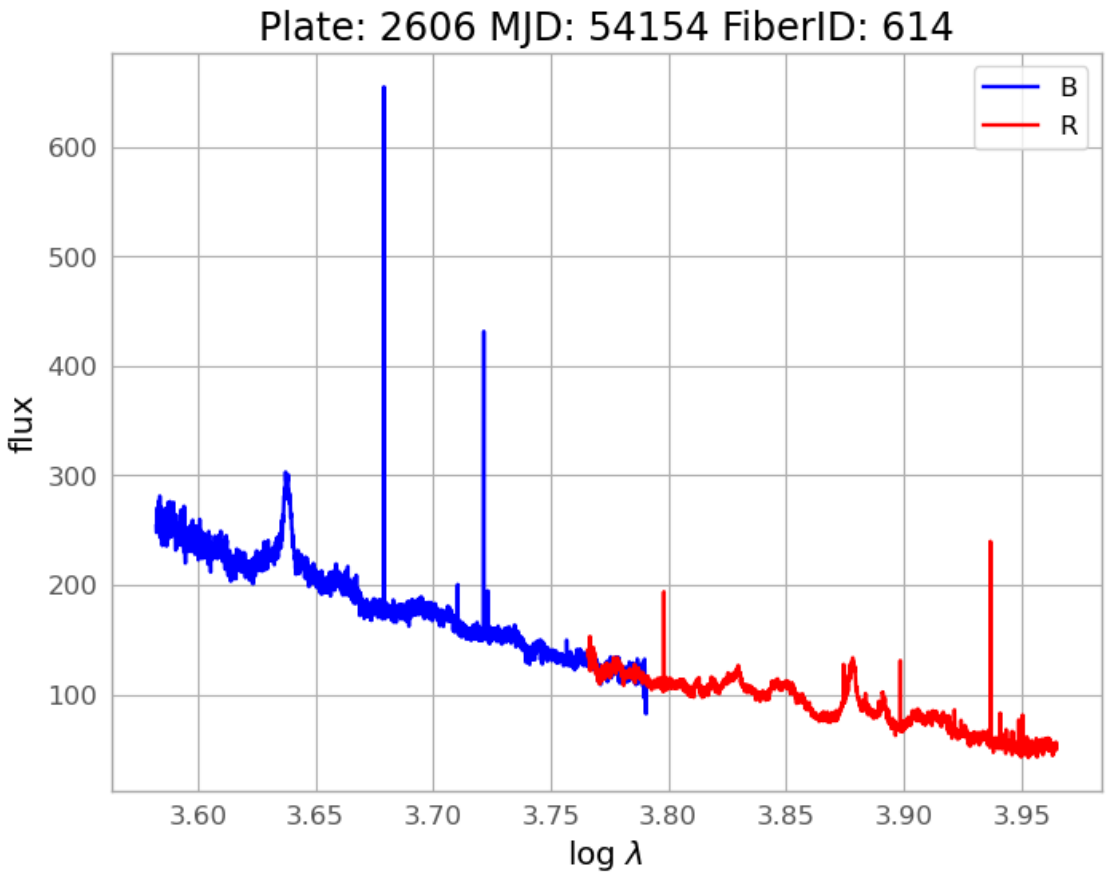


Plate: 2606 MJD: 54154 FiberID: 614
B
R
flux
600
500
400
300
200
100
3.60
3.65
3.70
3.75
3.80
3.85
3.90
3.95
log λ

39

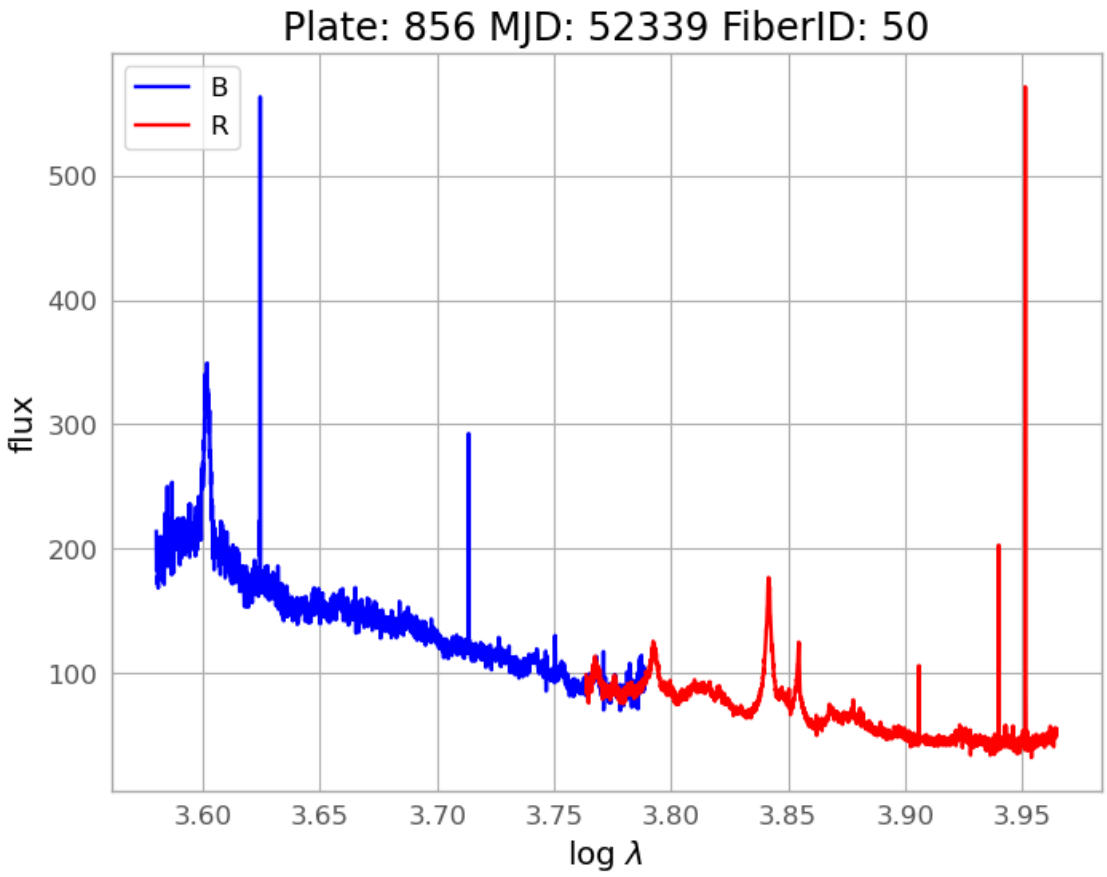


Plate: 856 MJD: 52339 FiberID: 50
B
R
flux
500
400
300
200
100
3.60
3.65
3.70
3.75
3.80
3.85
3.90
3.95
log λ

40

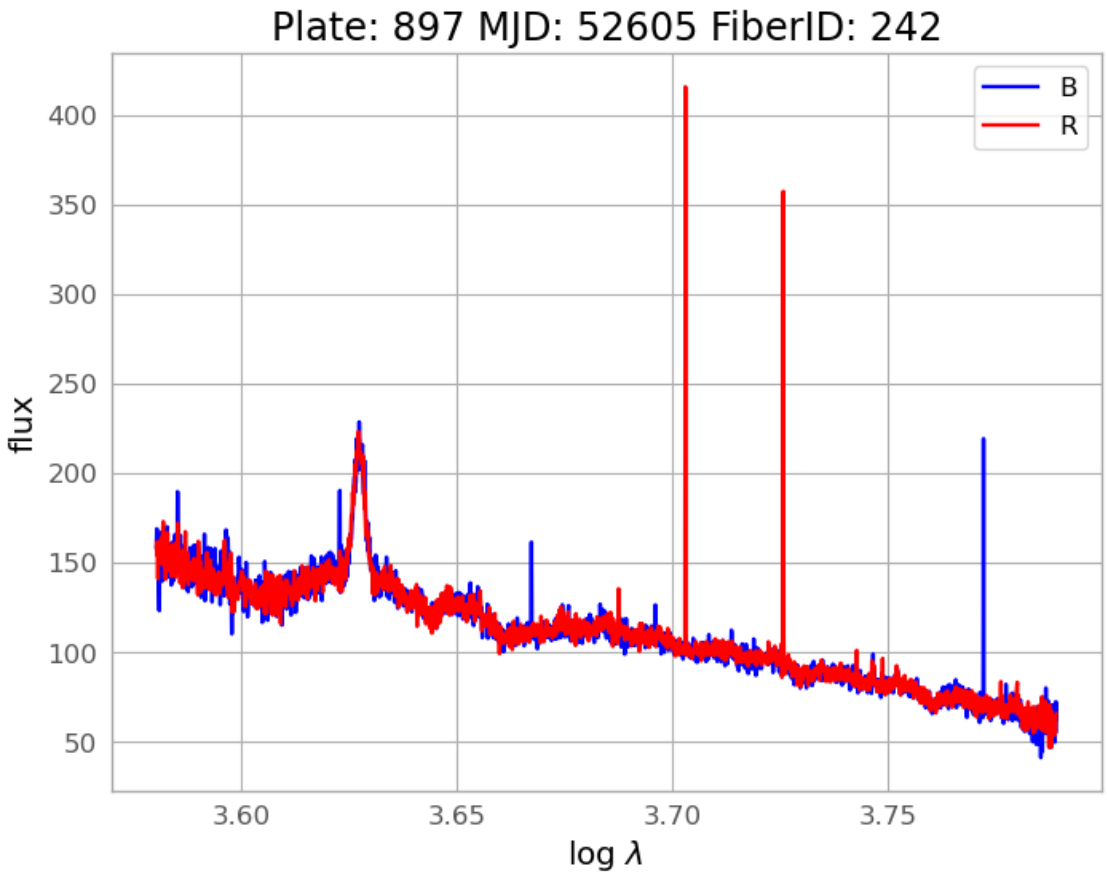


Plate: 897 MJD: 52605 FiberID: 242
B
R
flux
400
350
300
250
200
150
100
50
3.60
3.65
3.70
3.75
log λ

```
#  41
```

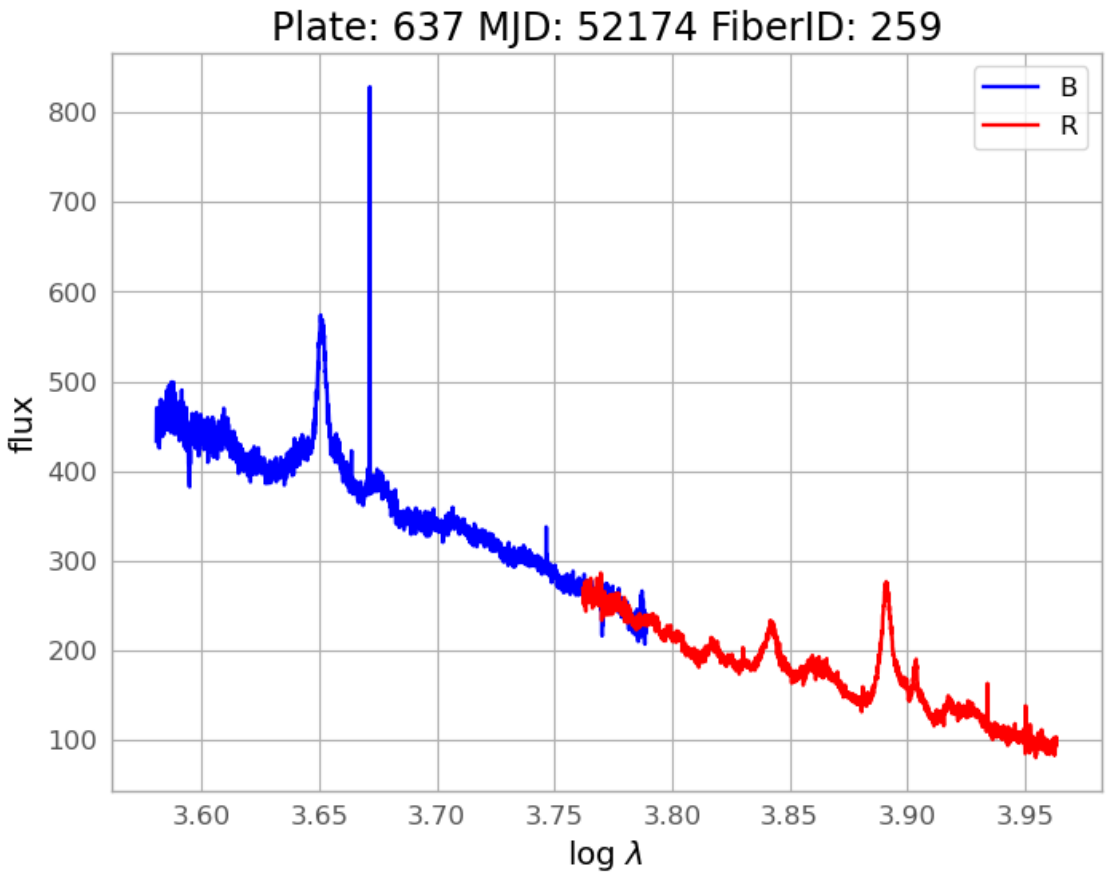


Plate: 637 MJD: 52174 FiberID: 259
B
R
flux
800
700
600
500
400
300
200
100
3.60
3.65
3.70
3.75
3.80
3.85
3.90
3.95
log λ

```
#  42
```

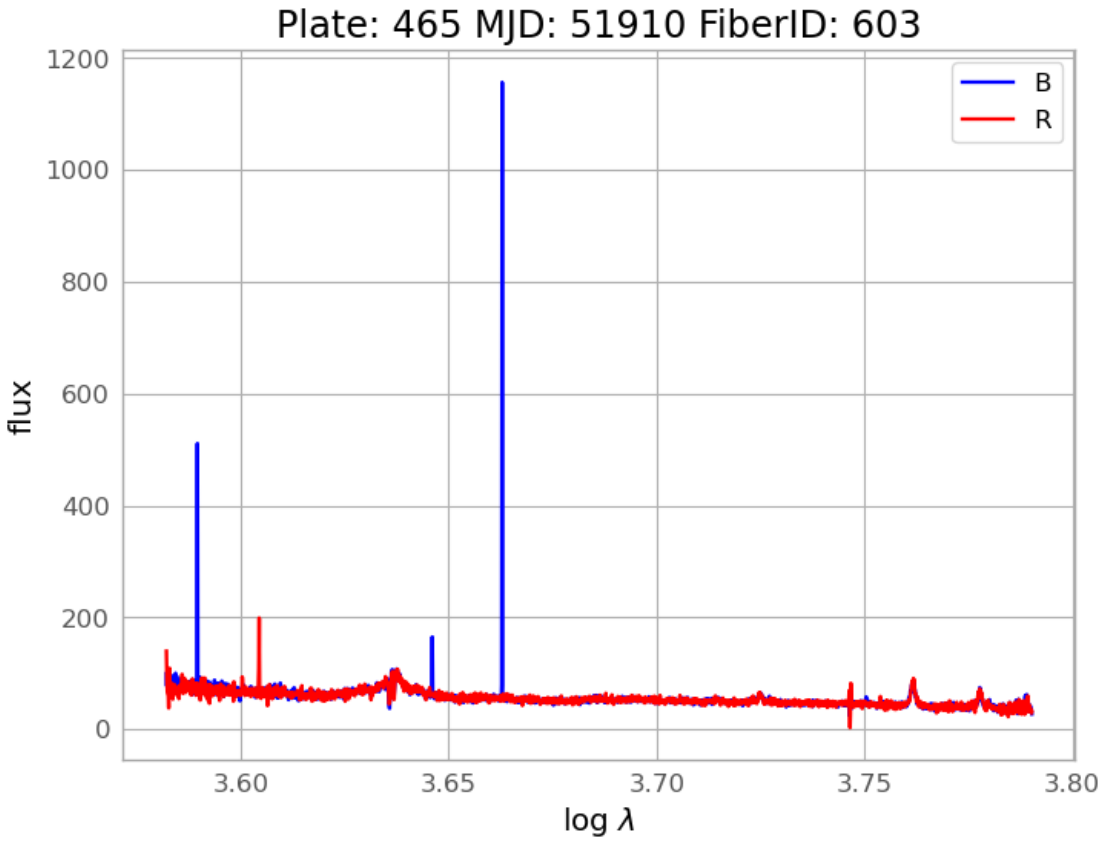


Plate: 465 MJD: 51910 FiberID: 603
B
R
flux
log λ
1200
1000
800
600
400
200
0
3.60
3.65
3.70
3.75
3.80

43

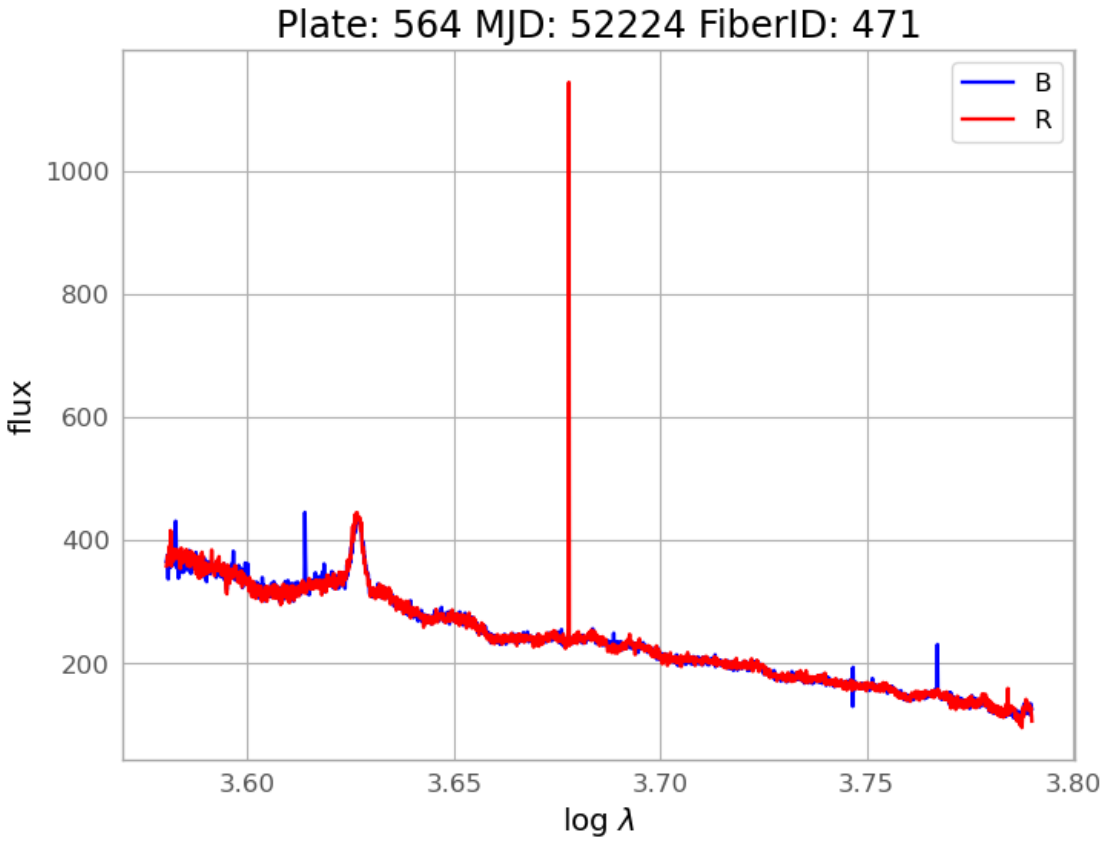


Plate: 564 MJD: 52224 FiberID: 471
B
R
flux
1000
800
600
400
200
3.60
3.65
3.70
3.75
3.80
log λ

```
#  44
```

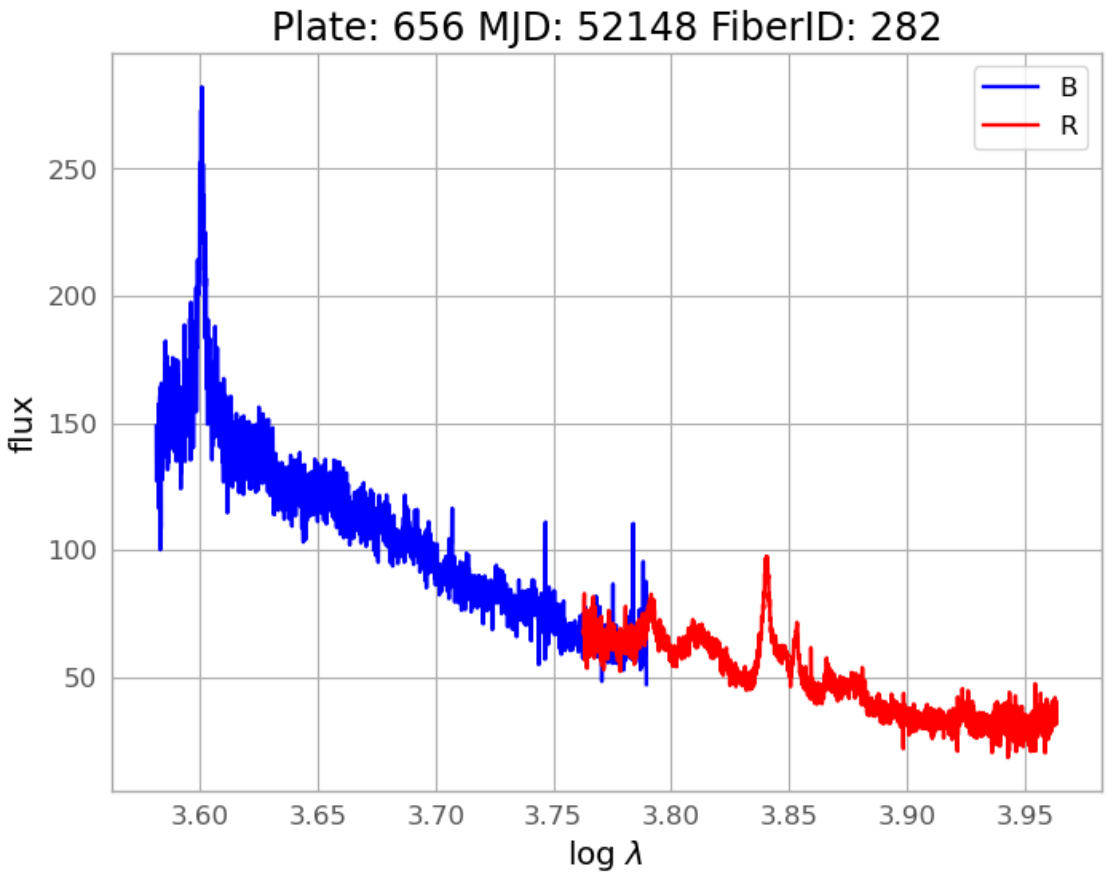


Plate: 656 MJD: 52148 FiberID: 282
B
R
flux
250
200
150
100
50
3.60
3.65
3.70
3.75
3.80
3.85
3.90
3.95
log λ

45

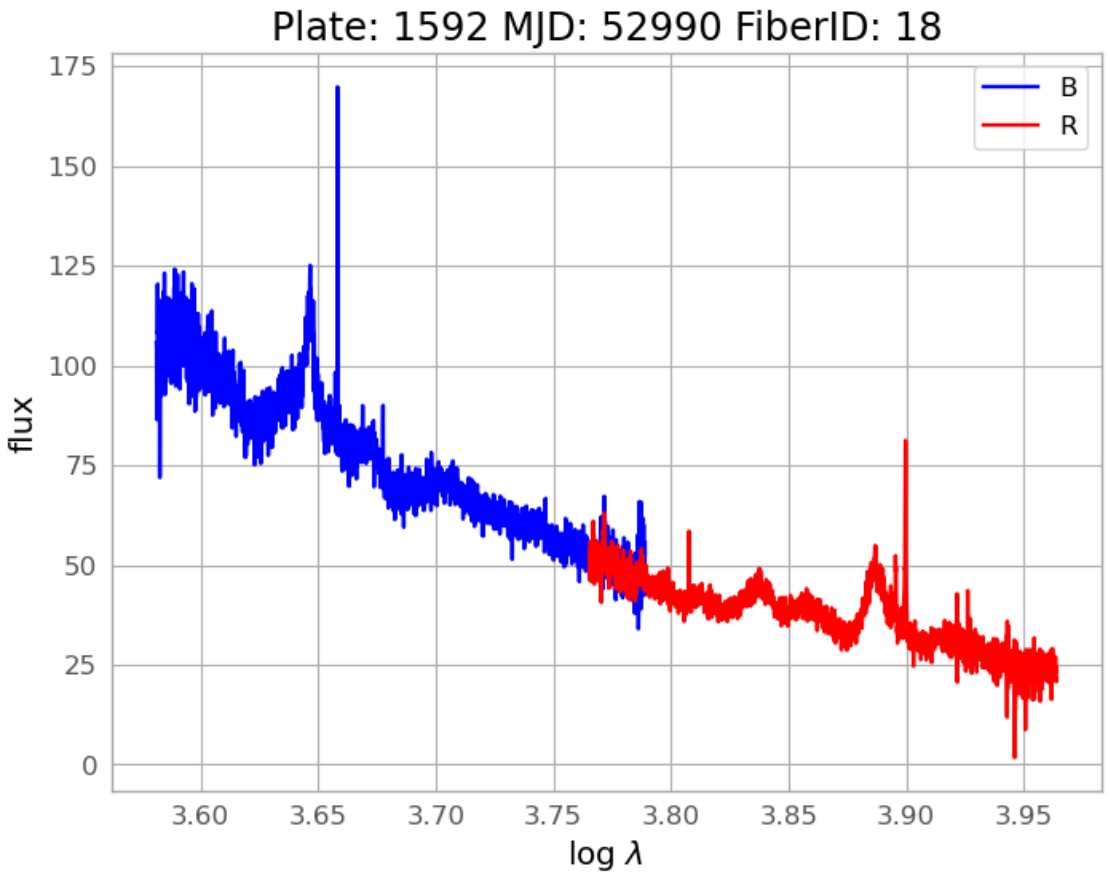


Plate: 1592 MJD: 52990 FiberID: 18
B
R
flux
log λ
175
150
125
100
75
50
25
0
3.60
3.65
3.70
3.75
3.80
3.85
3.90
3.95

46

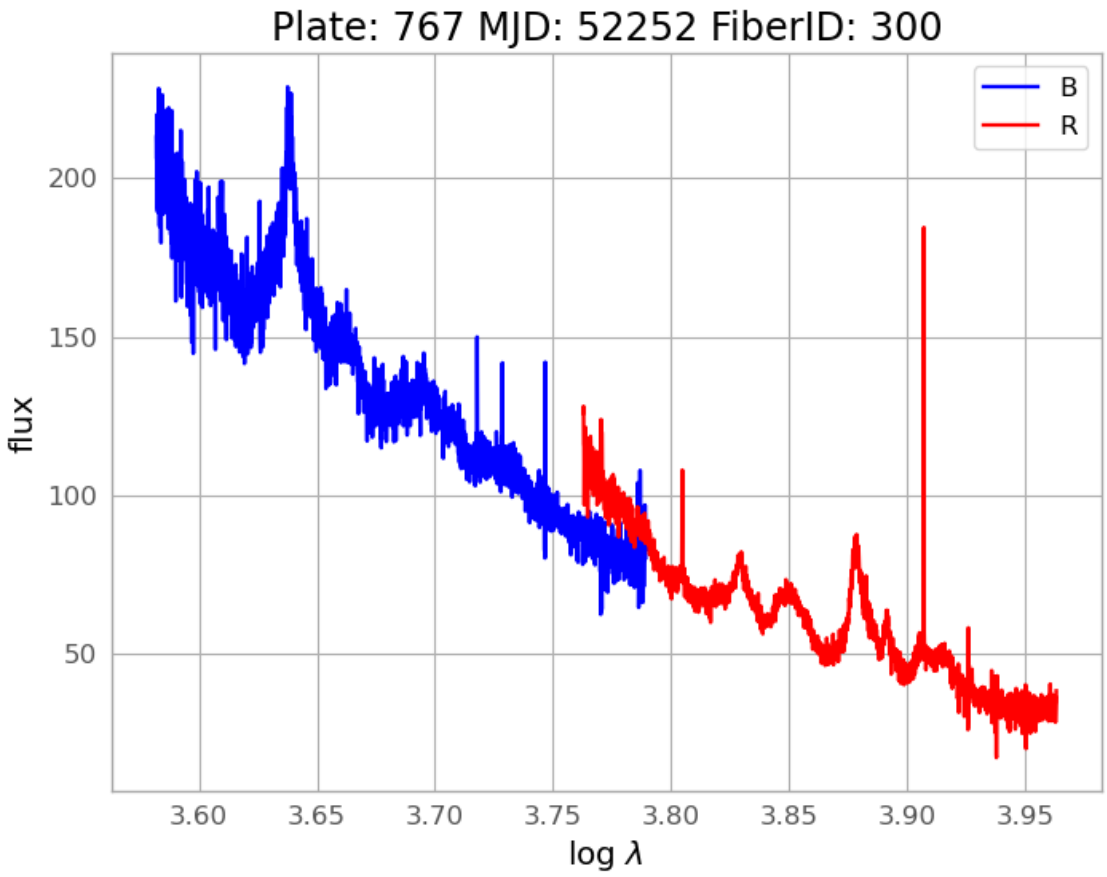
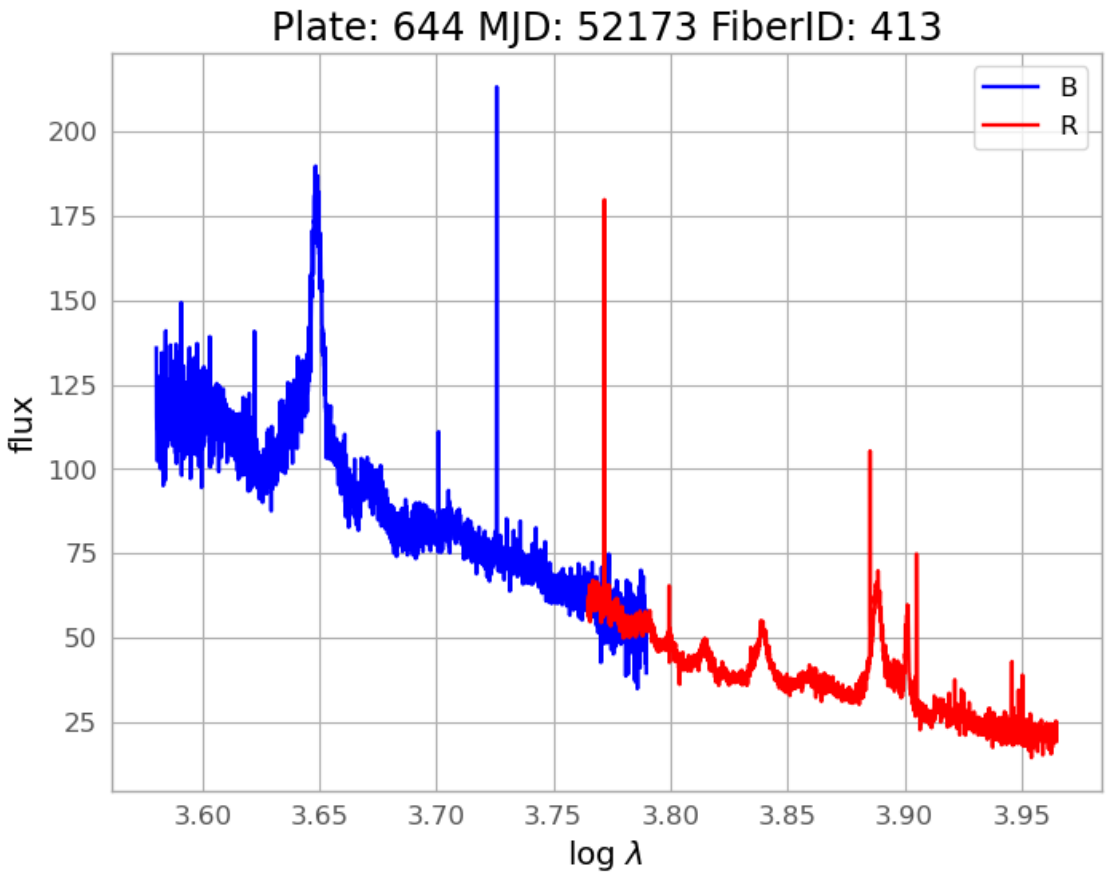


Plate: 767 MJD: 52252 FiberID: 300
B
R
flux
200
150
100
50
3.60
3.65
3.70
3.75
3.80
3.85
3.90
3.95
log λ

```
#  47
```



Note: From the results, we see 6 and 9 is not always the reduced result.

ii. AGNs Classification - THE FINAL PROJECT

In this part, we will look at classifications of emission-line galaxies using SDSS and WISE surveys.

But before that you need to know about different classifications of galaxies, so lets break them:

Emission Line Galaxies: Emission line galaxies are galaxies that exhibit prominent spectral emission lines in their spectra. These lines are caused by the excitation of atoms within the galaxies, often due to the presence of hot, ionized gas. The emission lines can originate from various processes such as star formation, active galactic nuclei (AGNs), or shocks from supernova remnants.

Active Galactic Nuclei (AGNs): AGNs are extremely luminous sources of radiation found at the centers of some galaxies. They are powered by the accretion of matter onto a supermassive black hole. AGNs exhibit various characteristics, including strong emission lines in their spectra, which can help distinguish them from other types of galaxies.

LINER (Low-Ionization Nuclear Emission-Line Region) Galaxies: LINER galaxies are a class of galaxies characterized by relatively weak emission lines originating from low-ionization species such as oxygen, sulfur, and nitrogen. The exact origin of the emission in LINERs is still debated, but it is often attributed to weak AGN activity or other processes such as shocks or hot, evolved stars.

Seyfert Galaxies: Seyfert galaxies are a type of galaxy that hosts an active nucleus with strong emission lines originating from highly ionized gas. They are characterized by their bright nuclei and are classified into two main types: Seyfert Type 1, which display broad emission lines, and Seyfert Type 2, which exhibit narrow emission lines.

BPT Diagram (Baldwin, Phillips, and Terlevich): The BPT diagram is a widely used diagnostic tool in astronomy, particularly in extragalactic astronomy, for distinguishing different types of ionizing sources within galaxies. It plots the flux ratios of specific emission lines, typically [O III] 5007/H versus [N II] 6583/H , or alternatively [O III] 5007/H versus [S II] 6716,6731/H . These emission lines are produced by various ionization mechanisms, and their ratios provide insights into the nature of the ionizing sources.

Kewley-Kauffmann Diagram: The Kewley-Kauffmann diagram is another tool used for galaxy classification, particularly for distinguishing between different types of AGNs and LINERs (Low-Ionization Nuclear Emission-line Regions).

Kewley Line: The Kewley-Kauffmann diagram includes a theoretical demarcation known as the "Kewley line," which separates star-forming galaxies from those hosting AGNs. This line is based on theoretical models of photoionization by AGNs and is designed to minimize contamination from star-forming galaxies. Galaxies lying above the Kewley line are classified as AGNs, while those below it are considered to be dominated by star formation.

Kauffmann Line: Another important feature of the Kewley-Kauffmann diagram is the "Kauffmann line," which is a empirical demarcation separating star-forming galaxies from LINERs. LINERs are galaxies characterized by low-ionization emission lines, typically arising from processes unrelated to AGNs, such as shocks or photoionization by older stars. Galaxies below the Kauffmann line are considered to be LINERs or composite objects where both star formation and AGN activity contribute to the emission lines.

[102]:
```
# imports

import pandas as pd
import numpy as np
from scipy.stats import gaussian_kde
import matplotlib
import matplotlib.pyplot as plt

from astroquery.sdss import SDSS
from calc_kcor import calc_kcor
```

0.1 TASK 1 - BPT & WHAN

[105]:
```
# sdss query through astropy

with open('query_task1.txt', 'r') as file:
    query = file.read().replace('\n', ' ') ## input is a single string

results = SDSS.query_sql(query, timeout=500, data_release=18).to_pandas()

results.to_csv('nlr.csv', sep=',')
```

```
C:\Users\Admin\Anaconda-Jupyter\Lib\site-packages\astroquery\sdss\core.py:874:
VisibleDeprecationWarning: Reading unicode strings without specifying the
encoding argument is deprecated. Set the encoding, use None for the system
default.
  arr = np.atleast_1d(np.genfromtxt(io.BytesIO(response.content),
```

[106]:
```
# our data

gal = pd.read_csv('nlr.csv', sep=',', index_col=0)
gal
```

[106]:

	plate	mjd	fiberid	class	subclass	z	\
0	266	51630	87	b'GALAXY'	b'STARBURST'	0.067414	
1	266	51602	159	b'GALAXY'	b'STARBURST'	0.053603	
2	267	51608	621	b'GALAXY'	b'STARFORMING'	0.035462	
3	269	51910	135	b'GALAXY'	b'STARBURST'	0.094104	
4	269	51910	288	b'GALAXY'	b'STARFORMING'	0.046238	
…	…	…	…	…	…	…	
29995	1056	52764	442	b'GALAXY'	b'STARFORMING'	0.065443	
29996	1057	52522	124	b'GALAXY'	b'STARFORMING'	0.132399	
29997	1057	52522	324	b'GALAXY'	b'STARFORMING'	0.027324	
29998	1059	52618	324	b'GALAXY'	b'STARFORMING'	0.061138	
29999	1060	52636	30	b'GALAXY'	b'AGN'	0.067646	

	h_alpha_flux	h_beta_flux	oiii_5007_flux	nii_6584_flux	oi_6300_flux	\
0	625.9243	117.55920	51.88348	187.59960	22.94478	
1	374.8391	82.36267	61.42209	105.32840	13.33986	
2	420.5194	106.90280	202.52370	88.06135	16.43069	
3	442.6733	135.58120	168.48460	94.99934	18.49554	
4	500.8737	113.95010	41.15973	158.55740	13.36059	

```
…            …          …              …              …             …
29995      242.2236    53.75721        28.80658       62.33384       8.12392
29996      224.5439    59.33580        61.03666       75.48510      11.15376
29997      133.1898    32.81077        13.54336       44.38828      10.53187
29998      661.8478   150.52940        72.86647      236.74930      26.05650
29999      869.6874   220.36940       822.69360      824.69550     157.90260

       sii_6717_flux  sii_6731_flux  h_alpha_eqw  nii_6584_eqw
0          125.94180       82.95983    -59.22600    -17.777280
1           79.49012       55.71835    -46.11523    -13.051050
2           94.87071       74.93069    -28.58870     -6.061126
3           87.43896       66.73173    -40.66494     -8.408584
4           99.77538       66.24239    -42.28374    -13.149500
…                …              …            …
29995       51.71253       34.32153    -30.26237     -7.933311
29996       43.88196       35.92858    -25.31144     -8.493839
29997       27.62878       20.86484    -14.10983     -4.689132
29998      140.26560       96.13190    -39.38181    -13.832420
29999      337.72890      284.01890    -10.15367     -8.948594

[30000 rows x 15 columns]
```

[107]:
```
gal.describe()
```

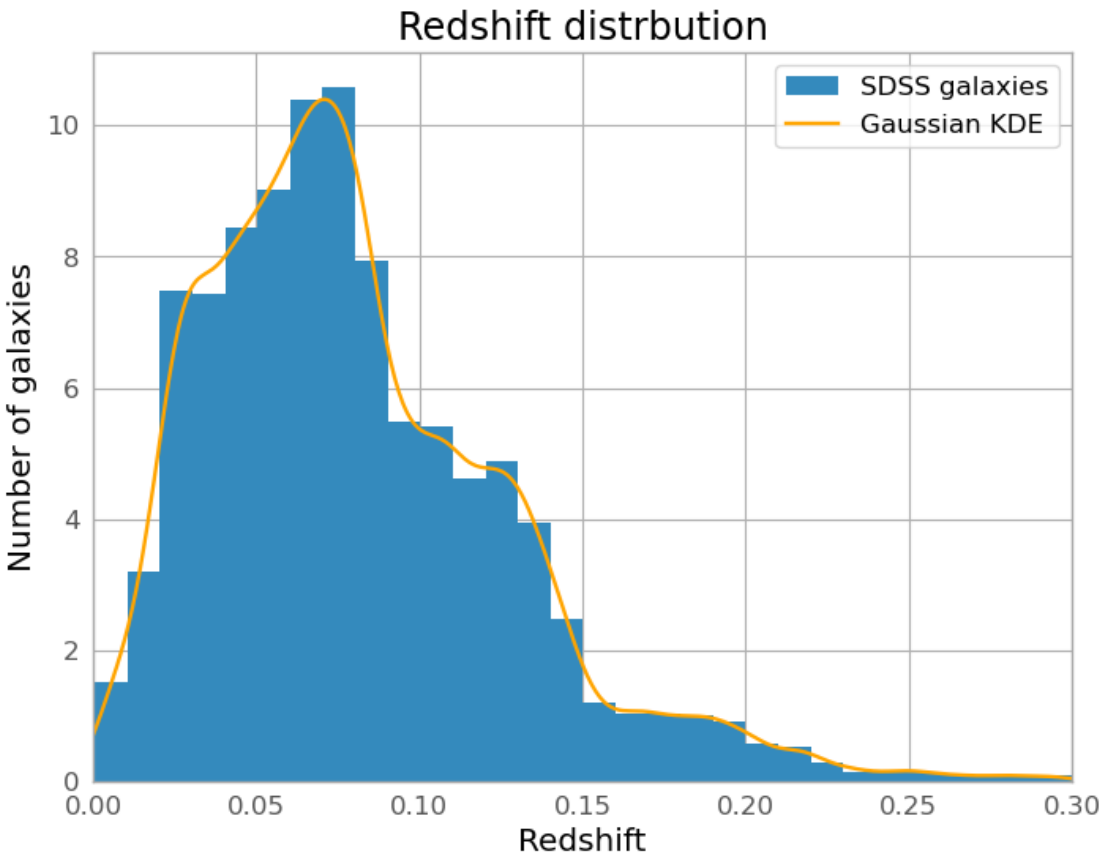
[107]:
```
              plate           mjd       fiberid             z  h_alpha_flux  \
count  30000.000000  30000.000000  30000.000000  30000.000000  3.000000e+04
mean     684.585967  52238.611533    323.547467      0.080721  3.058341e+07
std      415.696889    515.342175    184.012544      0.046703  1.719696e+09
min      266.000000  51602.000000      1.000000      0.000501  2.296743e+01
25%      393.000000  51883.000000    165.000000      0.046810  2.493693e+02
50%      521.000000  52054.000000    325.000000      0.072720  3.954241e+02
75%      878.000000  52427.000000    484.000000      0.107176  6.459334e+02
max     2213.000000  54567.000000    640.000000      0.299715  2.666877e+11

        h_beta_flux  oiii_5007_flux  nii_6584_flux  oi_6300_flux  \
count  3.000000e+04    3.000000e+04   3.000000e+04  3.000000e+04
mean   9.280430e+06    3.059827e+07   3.547983e+06  7.370555e+05
std    5.599923e+08    2.981135e+09   1.196292e+08  2.446583e+07
min    5.225145e+00    2.134984e+00   3.652907e+00  1.115707e+00
25%    6.166221e+01    3.359321e+01   7.134955e+01  1.034424e+01
50%    9.743359e+01    6.784680e+01   1.271078e+02  1.445737e+01
75%    1.593949e+02    1.741823e+02   2.200439e+02  2.208546e+01
max    8.824806e+10    5.018992e+11   7.924887e+09  2.096429e+09

       sii_6717_flux  sii_6731_flux   h_alpha_eqw  nii_6584_eqw
count   3.000000e+04   3.000000e+04  30000.000000  30000.000000
mean    3.335811e+06   2.364276e+06    -51.261392    -13.173843
```

```
std     1.046715e+08   7.393820e+07     69.550156      9.021134
min     9.068540e+00   5.725909e+00  -2195.250000   -283.320000
25%     5.082426e+01   3.648165e+01    -55.107388    -16.279055
50%     7.331984e+01   5.349077e+01    -36.750030    -11.235240
75%     1.106199e+02   8.183307e+01    -24.418220     -7.619751
max     6.167658e+09   4.302112e+09     -1.273633      2.317096
```

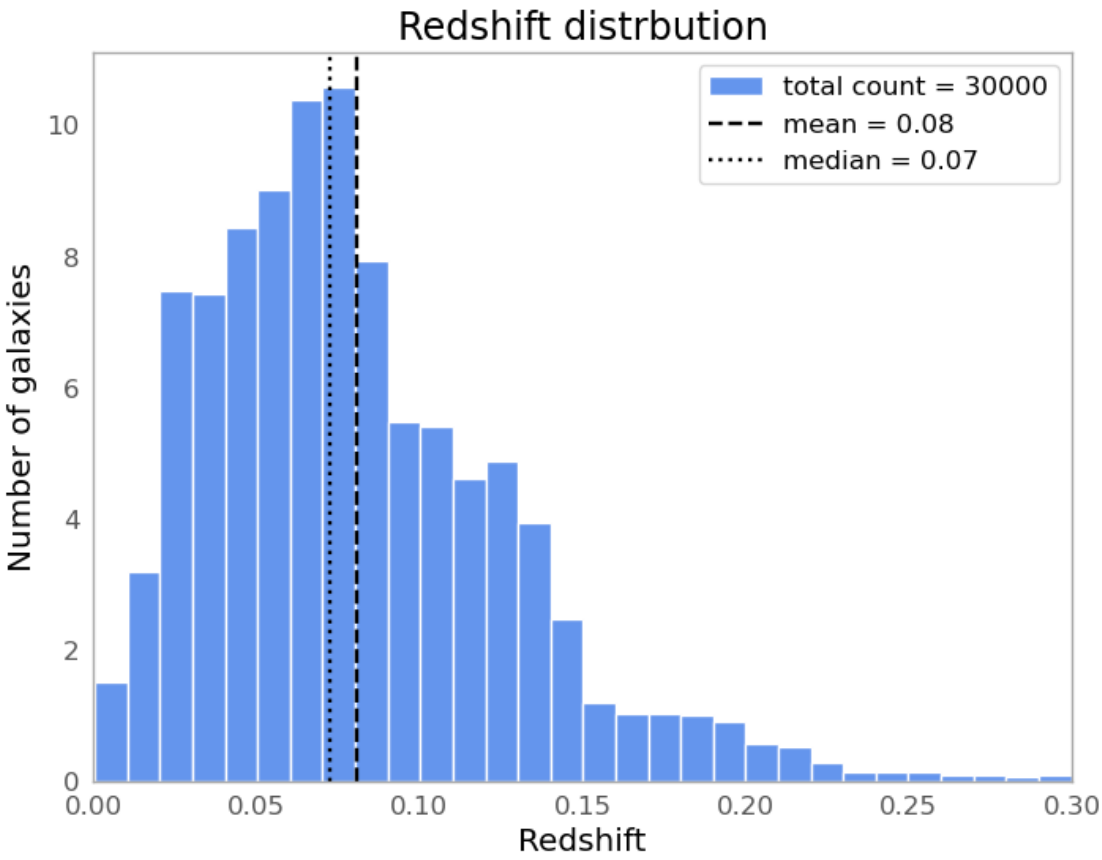
Exploring our data

[119]:
```
# redshift distribution

fig, ax = plt.subplots()
ax.set_xlim(0, 0.3)
gal.z.plot.hist(bins= 30, density=True, ax=ax, label="SDSS galaxies")
gal.z.plot.kde(ax=ax, label="Gaussian KDE", color='orange')
y = np.histogram(gal.z, bins= 30, density=True)
print(y[0])
plt.xlabel('Redshift')
plt.ylabel('Number of galaxies')
plt.title('Redshift distrbution')
plt.grid(True)
plt.legend()
plt.savefig('z_kde.pdf', dpi=100);
```

```
[ 1.52065179  3.21175025  7.48962781  7.41944388  8.44212398  9.02699005
 10.37719515 10.57772066  7.9441523   5.48103061  5.40750459  4.6254551
  4.88613826  3.94366837  2.46980587  1.19312679  1.04607474  1.03604847
  1.00596964  0.91239107  0.58820816  0.5313926   0.28407781  0.15707832
  0.14705204  0.15707832  0.11028903  0.09357857  0.0802102   0.09692066]
```



[120]:
```
# redshift distribution

fig, ax = plt.subplots()
ax.set_xlim(0, 0.3)
gal.z.plot.hist(bins= 30, color='cornflowerblue', edgecolor='white',␣
 ↪density=True, ax=ax, label=f"total count = {len(gal)}")
plt.axvline(np.mean(gal.z), linestyle='--', c='k', label=f"mean = {np.mean(gal.
 ↪z):.2f}")
plt.axvline(np.median(gal.z), linestyle=':', c='k', label=f"median = {np.
 ↪median(gal.z):.2f}")
plt.xlabel('Redshift')
plt.ylabel('Number of galaxies')
plt.title('Redshift distrbution')
plt.grid(False)
plt.legend()
plt.savefig('z_dist.pdf', dpi=100);
```



[121]:
```
# class distribution

gal['class'].value_counts()
```

[121]:
```
class
b'GALAXY'    29735
b'QSO'         265
Name: count, dtype: int64
```

[122]:
```
# subclass distribution

gal['subclass'].value_counts()
```

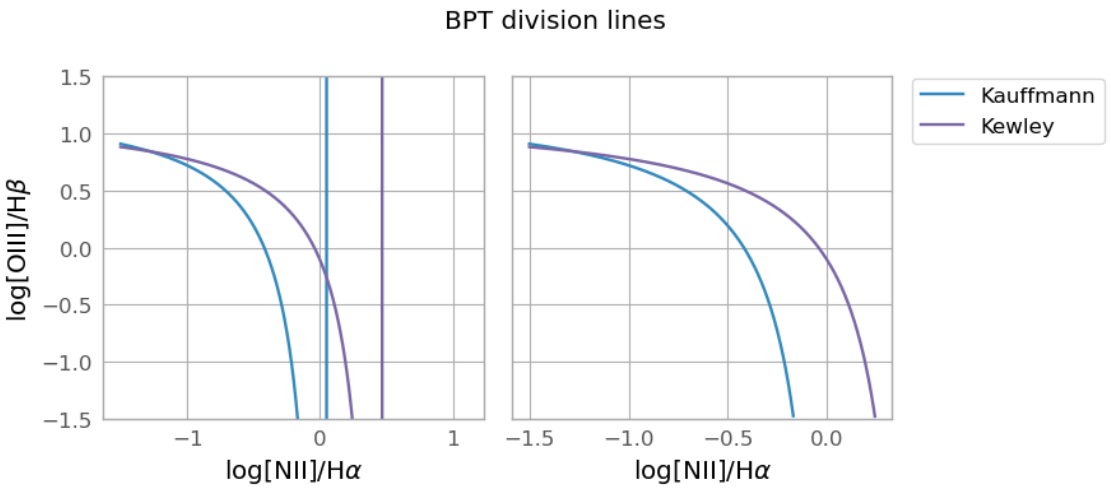
[122]:
```
subclass
b'STARFORMING'              13783
b'STARBURST'                12874
b'AGN'                       2008
b''                           896
b'AGN BROADLINE'              212
b'STARBURST BROADLINE'         97
```

```
b'BROADLINE'                  88
b'STARFORMING BROADLINE'      42
Name: count, dtype: int64
```

0.2 BPT diagram

[124]:
```
# BPT division lines

x = np.linspace(-1.5, 1.1, 1000)
y_kauffmann = 0.61/(x-0.05) + 1.3
y_kewley = 0.61/(x-0.47) + 1.19

fig, axs = plt.subplots(1,2, figsize=(9,4), sharey=True)

fig.suptitle("BPT division lines")

axs[0].plot(x, y_kauffmann, label="Kauffmann")
axs[0].plot(x, y_kewley, label="Kewley")
axs[0].set_ylim(-1.5, 1.5)
axs[0].set_xlabel(r"log[NII]/H$\alpha$")
axs[0].set_ylabel(r"log[OIII]/H$\beta$")

# division lines as a function to make it single valued in our limits

def kauffman(x):
    ## x lim was calculated for a y of -1.5
    return [0.61/(i-0.05) + 1.3 if (i < (0.61/(-1.5-1.3))+0.05) else -np.inf␣
 ↪for i in x]

def kewley(x):
   ## x lim was calculated for a y of -1.5
   return [0.61/(i-0.47) + 1.19 if (i < (0.61/(-1.5-1.19))+0.47) else -np.inf␣
 ↪for i in x]

axs[1].plot(x, kauffman(x), label="Kauffmann")
axs[1].plot(x, kewley(x), label="Kewley")
axs[1].set_ylim(-1.5, 1.5)
axs[1].set_xlabel(r"log[NII]/H$\alpha$")
axs[1].tick_params(direction='out')

for i in range(1,len(axs)):
    axs[i].set_ylim( axs[0].get_ylim() ) # align axes
    #axs[i].set_yticks([])

plt.subplots_adjust(wspace=.01)
plt.legend(bbox_to_anchor=(1.025,1.025), loc='upper left')
fig.tight_layout()
```

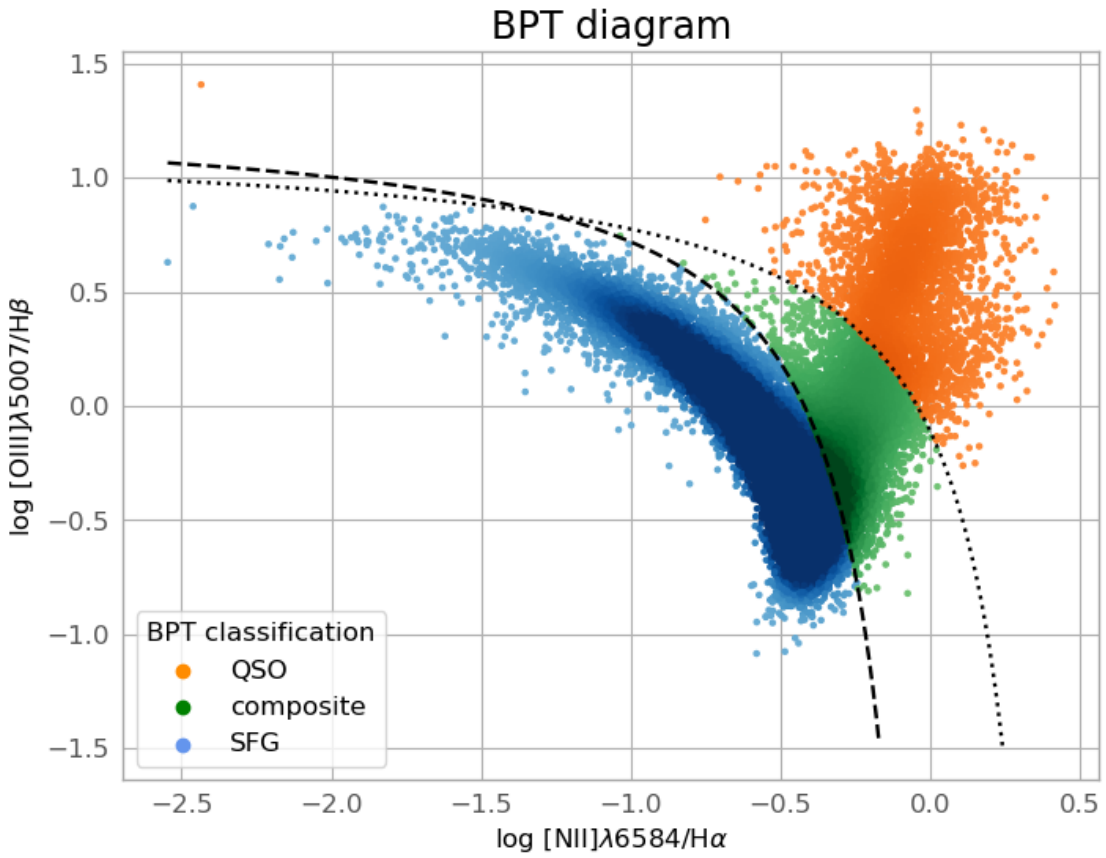
```
fig.savefig('bpt_division.pdf', dpi=100);
```



[125]:
```
# BPT classification

def bpt_classifier(df):
    df['x_BPT'] = np.log10(df['nii_6584_flux']/df['h_alpha_flux'])
    df['y_BPT'] = np.log10(df['oiii_5007_flux']/df['h_beta_flux'])
    df['class_Kauffmann'] = np.where((df['y_BPT'] <= kauffman(df['x_BPT'])),␣
 ↪"SFG", "QSO")
    df['class_Kewley'] = np.where((df['y_BPT'] <= kewley(df['x_BPT'])), "SFG",␣
 ↪"QSO")
    df['class_BPT'] = np.
 ↪where(((df['class_Kauffmann']=='SFG')&(df['class_Kewley']=='SFG')), "SFG",
                                    np.
 ↪where(((df['class_Kauffmann']=='QSO')&(df['class_Kewley']=='QSO')), "AGN",
                                            "composite"))
    return df

def bpt_plt(df):

    fig = plt.figure()

    x = df['x_BPT']
    y = df['y_BPT']
    xy = np.vstack([x,y])
    df["kde"] = gaussian_kde(xy)(xy)

    class_dict = {'SFG': plt.cm.Blues,
                  'composite': plt.cm.Greens,
```

```
                'AGN': plt.cm.Oranges
                }

    for class_gal in class_dict.keys():
        df_sub = df[df['class_BPT'] == class_gal]
        x_sub = df_sub['x_BPT']
        y_sub = df_sub['y_BPT']
        z_sub = df_sub['kde']
        normalize = matplotlib.colors.Normalize(vmin=-1, vmax=1)
        plt.scatter(x_sub, y_sub, c=z_sub, s=5, edgecolor=None,
                    cmap=class_dict[class_gal], norm=normalize)

    x = np.linspace(np.min(df['x_BPT']),np.max( df['x_BPT']), 1000)
    plt.plot(x, kauffman(x), 'k--', label="Kauffmann")
    plt.plot(x, kewley(x), 'k:', label="Kewley")

    scatter1 = plt.scatter([], [], c='cornflowerblue', label="SFG")
    scatter2 = plt.scatter([], [], c='green', label="composite")
    scatter3 = plt.scatter([], [], c='darkorange', label="AGN")
    plt.title('BPT diagram')
    plt.legend((scatter3, scatter2, scatter1), ["QSO", "composite", "SFG"],
               loc='lower left', title="BPT classification", fontsize='medium')
    plt.xlabel(r"log [NII]$\lambda$6584/H$\alpha$", fontsize=12)
    plt.ylabel(r"log [OIII]$\lambda$5007/H$\beta$", fontsize=12)
    plt.subplots_adjust(hspace=.0)

    return fig
```

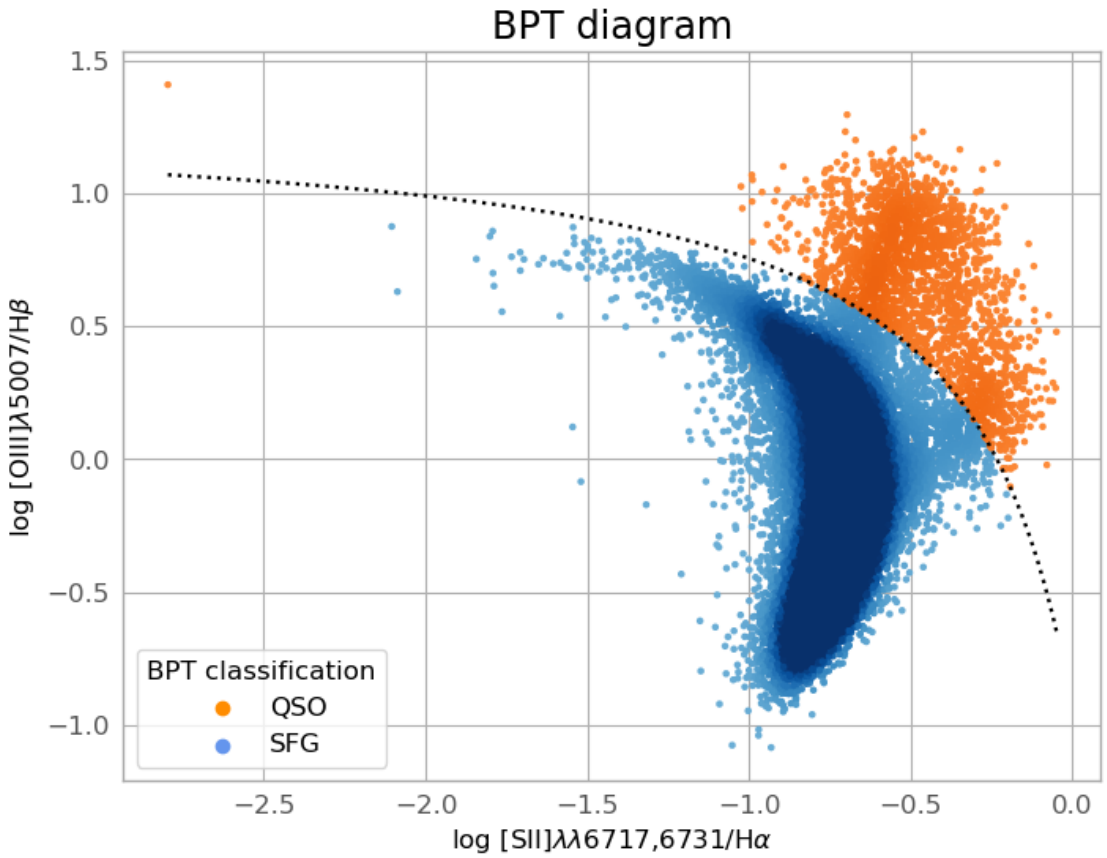
[126]:
```
fig = bpt_plt(bpt_classifier(gal))
fig.savefig('bpt_classic.pdf', dpi=100);
```



[127]:
```
# BPT  diagram - [OIII] 5007/H  vs.  [SII] 6716,6731/H

df=gal

# division line as a function to make it single valued in our limits

def kewley_si(x):
   ## x was calculated for a y of -1.5
   return [0.72/(i-0.32) + 1.30 if (i < (0.72/(-1.5-1.3))+0.32) else -np.inf␣
 ↪for i in x]

df['x_BPT'] = np.log10(df['sii_6717_flux']/df['h_alpha_flux'])
df['y_BPT'] = np.log10(df['oiii_5007_flux']/df['h_beta_flux'])
df['class_Kewley_si'] = np.where((df['y_BPT'] <= kewley_si(df['x_BPT'])),␣
 ↪"SFG", "QSO")

print(df['class_Kewley_si'].value_counts())

fig = plt.figure()
```

```
x = df['x_BPT']
y = df['y_BPT']
xy = np.vstack([x,y])
df["kde"] = gaussian_kde(xy)(xy)

class_dict = {'SFG': plt.cm.Blues,
              'QSO': plt.cm.Oranges
              }

for class_gal in class_dict.keys():
    df_sub = df[df['class_Kewley_si'] == class_gal]
    x_sub = df_sub['x_BPT']
    y_sub = df_sub['y_BPT']
    z_sub = df_sub['kde']
    normalize = matplotlib.colors.Normalize(vmin=-1, vmax=1)
    plt.scatter(x_sub, y_sub, c=z_sub, s=5, edgecolor=None,
                cmap=class_dict[class_gal], norm=normalize)

x = np.linspace(np.min(df['x_BPT']),np.max( df['x_BPT']), 1000)
plt.plot(x, kewley_si(x), 'k:', label="Kewley_si")

scatter1 = plt.scatter([], [], c='cornflowerblue', label="SFG")
scatter3 = plt.scatter([], [], c='darkorange', label="QSO")
plt.title('BPT diagram')
plt.legend((scatter3, scatter1), ["QSO", "SFG"],
           loc='lower left', title="BPT classification", fontsize='medium')
plt.xlabel(r"log [SII]$\lambda$$\lambda$6717,6731/H$\alpha$", fontsize=12)
plt.ylabel(r"log [OIII]$\lambda$5007/H$\beta$", fontsize=12)
plt.subplots_adjust(hspace=.0)

fig.savefig("bpt_oiii_sii.pdf", dpi=100);
```

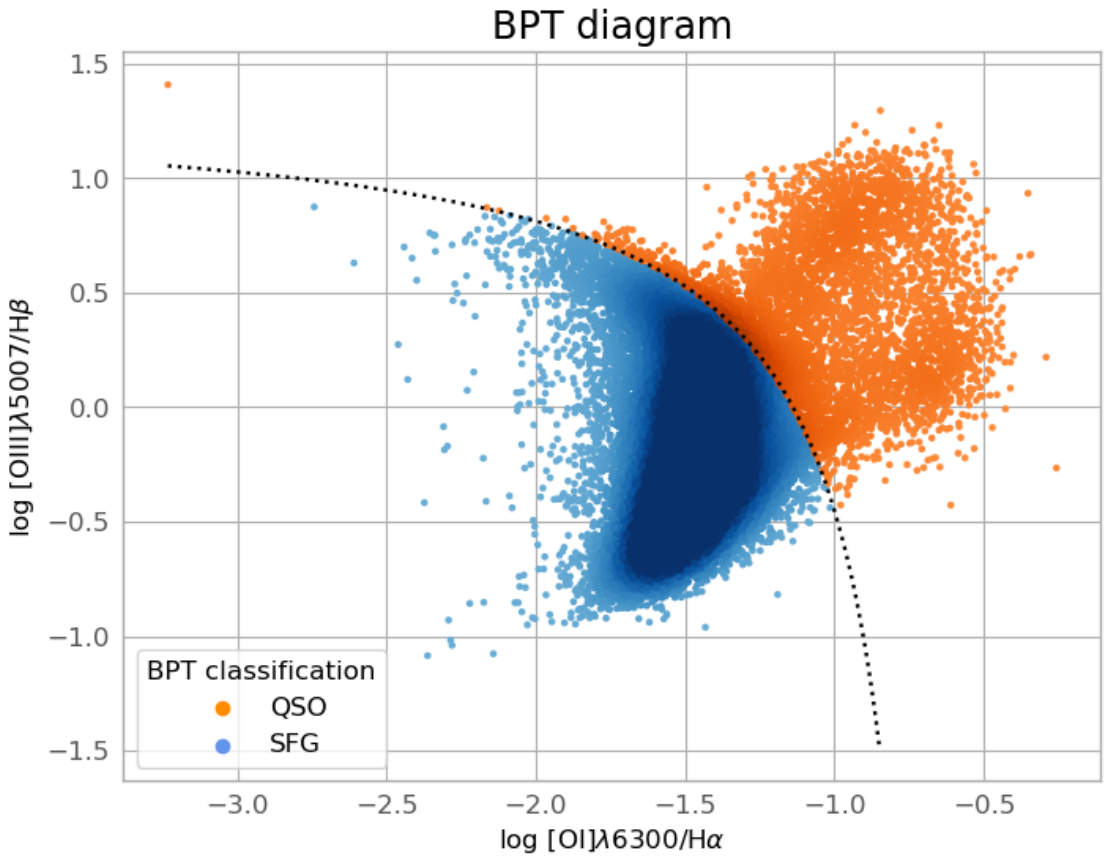
```
class_Kewley_si
SFG    28016
QSO     1984
Name: count, dtype: int64
```



[128]:
```
# BPT  diagram - [OIII] 5007/H  vs.  [OI] 6300/H

df=gal

# division line as a function to make it single valued in our limits

def kewley_oi(x):
   ## x was calculated for a y of -1.5
   return [0.73/(i+0.59) + 1.33 if (i < (0.73/(-1.5-1.33))-0.59) else -np.inf␣
 ↪for i in x]

df['x_BPT'] = np.log10(df['oi_6300_flux']/df['h_alpha_flux'])
df['y_BPT'] = np.log10(df['oiii_5007_flux']/df['h_beta_flux'])
df['class_Kewley_oi'] = np.where((df['y_BPT'] <= kewley_oi(df['x_BPT'])),␣
 ↪"SFG", "QSO")

print(df['class_Kewley_oi'].value_counts())

fig = plt.figure()
```

```
x = df['x_BPT']
y = df['y_BPT']
xy = np.vstack([x,y])
df["kde"] = gaussian_kde(xy)(xy)

class_dict = {'SFG': plt.cm.Blues,
              'QSO': plt.cm.Oranges
              }

for class_gal in class_dict.keys():
    df_sub = df[df['class_Kewley_oi'] == class_gal]
    x_sub = df_sub['x_BPT']
    y_sub = df_sub['y_BPT']
    z_sub = df_sub['kde']
    normalize = matplotlib.colors.Normalize(vmin=-1, vmax=1)
    plt.scatter(x_sub, y_sub, c=z_sub, s=5, edgecolor=None,
                cmap=class_dict[class_gal], norm=normalize)

x = np.linspace(np.min(df['x_BPT']),np.max( df['x_BPT']), 1000)
plt.plot(x, kewley_oi(x), 'k:', label="Kewley_oi")

scatter1 = plt.scatter([], [], c='cornflowerblue', label="SFG")
scatter3 = plt.scatter([], [], c='darkorange', label="QSO")
plt.title('BPT diagram')
plt.legend((scatter3, scatter1), ["QSO", "SFG"],
           loc='lower left', title="BPT classification", fontsize='medium')
plt.xlabel(r"log [OI]$\lambda$6300/H$\alpha$", fontsize=12)
plt.ylabel(r"log [OIII]$\lambda$5007/H$\beta$", fontsize=12)
plt.subplots_adjust(hspace=.0)

fig.savefig("bpt_oiii_oi.pdf", dpi=100);
```

```
class_Kewley_oi
SFG    25649
QSO     4351
Name: count, dtype: int64
```



0.3 WHAN Diagram

[132]:
```
# WHAN classification

gal = bpt_classifier(gal)

gal['y_WHAN'] = np.abs(gal['h_alpha_eqw'])
gal['z_WHAN'] = np.abs(gal['nii_6584_eqw'])

gal['group_WHAN'] = np.where(((gal['z_WHAN']>0.5) & (gal['y_WHAN']>0.5)),␣
 ↪"ELG", "lineless")

for i in gal.index:
    if gal.loc[i, 'y_WHAN'] > 0.5:
        if gal.loc[i, 'group_WHAN'] != "lineless":
            if gal.loc[i, 'y_WHAN'] > 3:
                if gal.loc[i, 'x_BPT'] > -0.4:
                    if gal.loc[i, 'y_WHAN'] > 6:
                        gal.loc[i, 'subclass_WHAN'] = "Seyfert"
```

```
                        gal.loc[i, 'class_WHAN'] = "AGN"
                    else:
                        gal.loc[i, 'subclass_WHAN'] = "wAGN"
                        gal.loc[i, 'class_WHAN'] = "LINER"
                else:
                    gal.loc[i, 'subclass_WHAN'] = "SFG"
                    gal.loc[i, 'class_WHAN'] = "SFG"
            else:
                gal.loc[i, 'subclass_WHAN'] = "RG"
                gal.loc[i, 'class_WHAN'] = "LINER"
        else:
            gal.loc[i, 'subclass_WHAN'] = "uncertain"
            gal.loc[i, 'class_WHAN'] = "uncertain"
    else:
        gal.loc[i, 'subclass_WHAN'] = "passive"
```

[133]:
```
gal['group_WHAN'].value_counts()
```

[133]:
```
group_WHAN
ELG         29991
lineless        9
Name: count, dtype: int64
```

[134]:
```
gal['class_WHAN'].value_counts()
```

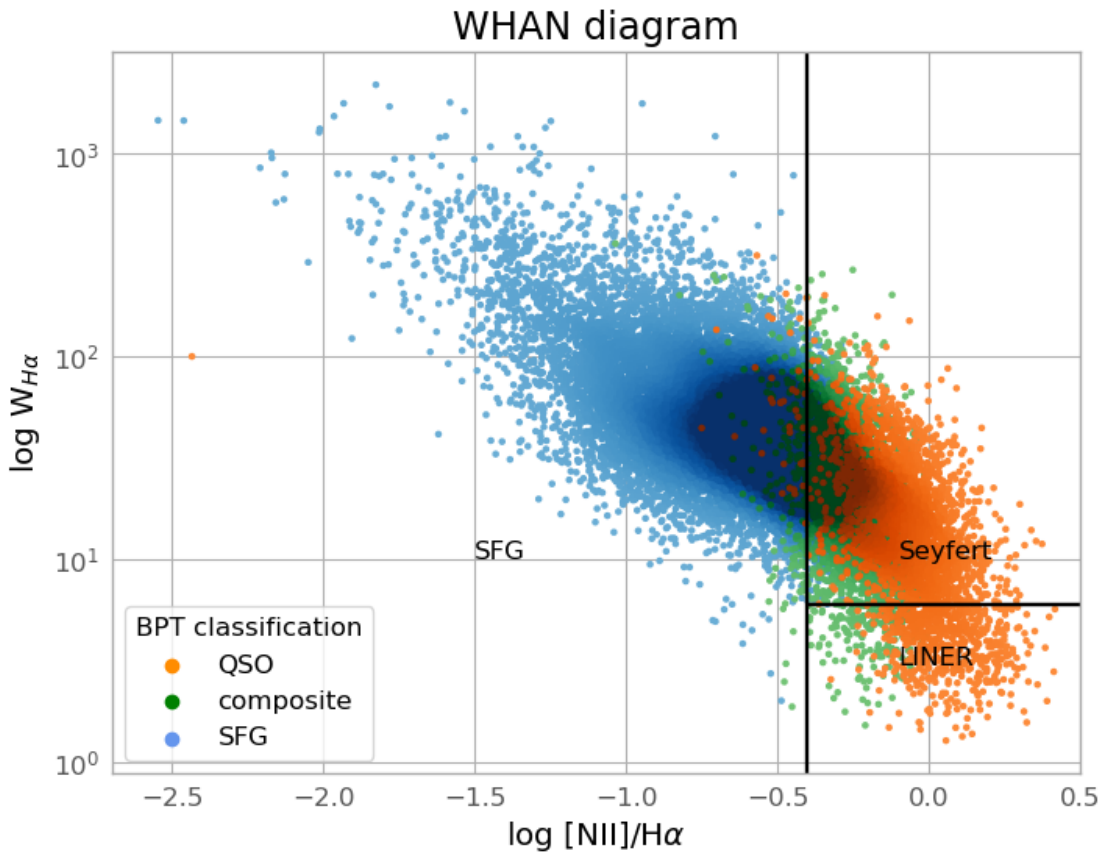
[134]:
```
class_WHAN
SFG          20910
AGN           8291
LINER          790
uncertain        9
Name: count, dtype: int64
```

[135]:
```
gal['subclass_WHAN'].value_counts()
```

[135]:
```
subclass_WHAN
SFG          20910
Seyfert       8291
wAGN           615
RG             175
uncertain        9
Name: count, dtype: int64
```

[136]:
```
# WHAN diagram

df = gal
x = df['x_BPT']
y = np.log10(df['y_WHAN'])
```

```
xy = np.vstack([x,y])
df['kde'] = gaussian_kde(xy)(xy)

class_dict = {'SFG': plt.cm.Blues,
              'composite': plt.cm.Greens,
              'AGN': plt.cm.Oranges
              }

for class_gal in class_dict.keys():
    df_sub = df[df['class_BPT'] == class_gal]
    x_sub = df_sub['x_BPT']
    y_sub = df_sub['y_WHAN']
    z_sub = df_sub['kde']
    normalize = matplotlib.colors.Normalize(vmin=-1, vmax=1)
    plt.scatter(x_sub, y_sub, c=z_sub, s=5, edgecolor=None,
                cmap=class_dict[class_gal], norm=normalize)

plt.axvline(x=-0.4, color='black')
plt.hlines(y=6, xmin=-0.4, xmax=0.5, color='black')
plt.annotate('Seyfert', xy=(-0.1,10), color='black')
plt.annotate('LINER', xy=(-0.1,3), color='black')
plt.annotate('SFG', xy=(-1.5,10), color='black')
plt.yscale('log')
plt.xlim(right=0.5)

scatter1 = plt.scatter([], [], c='cornflowerblue', label="SFG")
scatter2 = plt.scatter([], [], c='green', label="composite")
scatter3 = plt.scatter([], [], c='darkorange', label="AGN")
plt.legend((scatter3, scatter2, scatter1), ["QSO", "composite", "SFG"],
           loc='lower left', title="BPT classification", fontsize='medium')

plt.xlabel(r"log [NII]/H$\alpha$")
plt.ylabel(r"log W$_{H\alpha}$")
plt.title("WHAN diagram")
plt.savefig('whan.pdf', dpi=100);
```

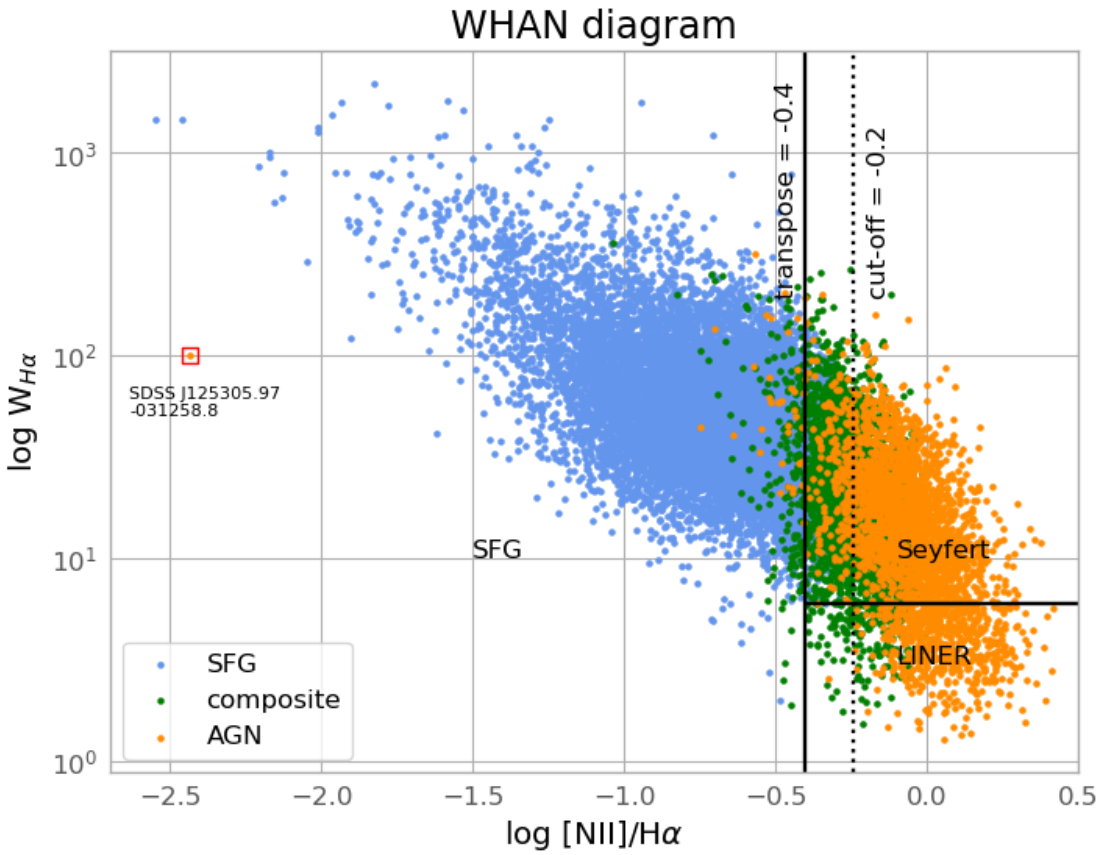


[137]:
```
# WHAN diagram

df = gal

class_dict = {'SFG': 'cornflowerblue',
              'composite': 'green',
              'AGN': 'darkorange'
              }

for class_gal in class_dict.keys():
    df_sub = df[df['class_BPT'] == class_gal]
    x_sub = df_sub['x_BPT']
    y_sub = df_sub['y_WHAN']
    plt.scatter(x_sub, y_sub, c=class_dict[class_gal], s=5, label=class_gal)
    if class_gal=='SFG':
        plt.axvline(np.max(df_sub['x_BPT']), ls=':', color='black')
        plt.annotate(f"cut-off = {np.max(df_sub['x_BPT']):.1f}", xy=(np.
 ↪max(df_sub['x_BPT'])+0.05,200), rotation=90, color='black')
```

```
plt.axvline(x=-0.4, color='black')
plt.annotate("transpose = -0.4", xy=(-0.4-0.1,200), rotation=90, color='black')
plt.hlines(y=6, xmin=-0.4, xmax=0.5, color='black')
plt.annotate("Seyfert", xy=(-0.1,10), color='black')
plt.annotate("LINER", xy=(-0.1,3), color='black')
plt.annotate("SFG", xy=(-1.5,10), color='black')
plt.yscale('log')
plt.xlim(right=0.5)

#df_sub = df[df['group_WHAN'] == 'lineless']
#plt.scatter(df_sub['x_BPT'], df_sub['y_WHAN'], ec='red', fc='None',␣
 ↪marker='o', s=50, label='uncertain')

df_sub = df[(df.class_BPT =='AGN') & (df.subclass_WHAN=='SFG') & (df.x_BPT<-2)]
plt.scatter(df_sub['x_BPT'], df_sub['y_WHAN'], ec='red', fc='None', marker='s',␣
 ↪s=50)
plt.annotate('SDSS J125305.97\n-031258.8', xy=(df_sub['x_BPT']-0.2,␣
 ↪df_sub['y_WHAN']-50), color='black', fontsize=8)

plt.xlabel(r"log [NII]/H$\alpha$")
plt.ylabel(r"log W$_{H\alpha}$")
plt.title("WHAN diagram")
plt.legend()
plt.savefig('whan_annotated.pdf', dpi=100);
```

```
C:\Users\Admin\Anaconda-Jupyter\Lib\site-packages\matplotlib\text.py:1461:
FutureWarning: Calling float on a single element Series is deprecated and will
raise a TypeError in the future. Use float(ser.iloc[0]) instead
  x = float(self.convert_xunits(x))
C:\Users\Admin\Anaconda-Jupyter\Lib\site-packages\matplotlib\text.py:1463:
FutureWarning: Calling float on a single element Series is deprecated and will
raise a TypeError in the future. Use float(ser.iloc[0]) instead
  y = float(self.convert_yunits(y))
C:\Users\Admin\Anaconda-Jupyter\Lib\site-packages\matplotlib\text.py:753:
FutureWarning: Calling float on a single element Series is deprecated and will
raise a TypeError in the future. Use float(ser.iloc[0]) instead
  posx = float(self.convert_xunits(self._x))
C:\Users\Admin\Anaconda-Jupyter\Lib\site-packages\matplotlib\text.py:754:
FutureWarning: Calling float on a single element Series is deprecated and will
raise a TypeError in the future. Use float(ser.iloc[0]) instead
  posy = float(self.convert_yunits(self._y))
C:\Users\Admin\Anaconda-Jupyter\Lib\site-packages\matplotlib\text.py:1461:
FutureWarning: Calling float on a single element Series is deprecated and will
raise a TypeError in the future. Use float(ser.iloc[0]) instead
  x = float(self.convert_xunits(x))
C:\Users\Admin\Anaconda-Jupyter\Lib\site-packages\matplotlib\text.py:1463:
FutureWarning: Calling float on a single element Series is deprecated and will
```

```
raise a TypeError in the future. Use float(ser.iloc[0]) instead
  y = float(self.convert_yunits(y))
C:\Users\Admin\Anaconda-Jupyter\Lib\site-packages\matplotlib\text.py:753:
FutureWarning: Calling float on a single element Series is deprecated and will
raise a TypeError in the future. Use float(ser.iloc[0]) instead
  posx = float(self.convert_xunits(self._x))
C:\Users\Admin\Anaconda-Jupyter\Lib\site-packages\matplotlib\text.py:754:
FutureWarning: Calling float on a single element Series is deprecated and will
raise a TypeError in the future. Use float(ser.iloc[0]) instead
  posy = float(self.convert_yunits(self._y))
C:\Users\Admin\Anaconda-Jupyter\Lib\site-packages\matplotlib\text.py:894:
FutureWarning: Calling float on a single element Series is deprecated and will
raise a TypeError in the future. Use float(ser.iloc[0]) instead
  x = float(self.convert_xunits(self._x))
C:\Users\Admin\Anaconda-Jupyter\Lib\site-packages\matplotlib\text.py:895:
FutureWarning: Calling float on a single element Series is deprecated and will
raise a TypeError in the future. Use float(ser.iloc[0]) instead
  y = float(self.convert_yunits(self._y))
```



[138]:
```
gal[(gal.class_WHAN =='uncertain')]
```

[138]:

	plate	mjd	fiberid	class	subclass	z	h_alpha_flux	\
2786	396	51816	237	b'GALAXY'	b'STARBURST'	0.100061	196.14560	
2905	399	51817	355	b'GALAXY'	b''	0.012659	63.95274	
3378	519	52283	165	b'GALAXY'	b'STARBURST'	0.100150	160.69910	
5664	678	52884	549	b'GALAXY'	b''	0.032510	1618.49400	
5778	784	52327	332	b'GALAXY'	b'STARBURST'	0.258647	365.55470	
11081	1086	52525	39	b'GALAXY'	b'STARBURST'	0.172050	200.86410	
19362	424	51893	124	b'GALAXY'	b''	0.051915	342.23110	
26084	691	52199	594	b'GALAXY'	b''	0.193502	149.86440	
26101	709	52205	79	b'GALAXY'	b'STARBURST'	0.029939	264.30350	

	h_beta_flux	oiii_5007_flux	nii_6584_flux	…	class_Kewley	\
2786	49.47662	35.49763	63.273140	…	SFG	
2905	16.64981	17.17562	15.545620	…	SFG	
3378	41.24368	21.36432	37.753170	…	SFG	
5664	339.11660	248.90660	491.326400	…	SFG	
5778	104.81580	157.18580	62.712300	…	SFG	
11081	49.41750	75.75452	38.704350	…	SFG	
19362	105.00790	241.47890	39.650960	…	SFG	
26084	33.99573	40.64509	32.656760	…	SFG	
26101	96.82309	195.83810	6.336245	…	SFG	

	class_BPT	kde	class_Kewley_si	class_Kewley_oi	y_WHAN	\
2786	SFG	4.175918	SFG	SFG	41.128430	
2905	SFG	0.003629	SFG	QSO	3.830764	
3378	SFG	1.892396	SFG	SFG	36.970360	
5664	SFG	2.910932	SFG	SFG	52.966190	
5778	SFG	0.599071	SFG	SFG	85.894600	
11081	SFG	1.245179	SFG	SFG	52.086310	
19362	SFG	0.522658	SFG	SFG	51.211990	
26084	SFG	1.470725	SFG	QSO	56.620000	
26101	SFG	0.002227	SFG	SFG	41.393830	

	z_WHAN	group_WHAN	subclass_WHAN	class_WHAN
2786	0.266877	lineless	uncertain	uncertain
2905	0.449749	lineless	uncertain	uncertain
3378	0.373773	lineless	uncertain	uncertain
5664	0.099646	lineless	uncertain	uncertain
5778	0.297303	lineless	uncertain	uncertain
11081	0.101586	lineless	uncertain	uncertain
19362	0.433846	lineless	uncertain	uncertain
26084	0.143819	lineless	uncertain	uncertain
26101	0.450224	lineless	uncertain	uncertain

[9 rows x 28 columns]

[139]:
```
gal[(gal.class_BPT =='AGN') & (gal.class_WHAN=='SFG') & (gal.x_BPT<-2)]
```

[139]:
```
       plate    mjd  fiberid      class      subclass         z  h_alpha_flux  \
12353    337  51997       97  b'GALAXY'  b'STARBURST'  0.022723     1352901.0

       h_beta_flux  oiii_5007_flux  nii_6584_flux  …  class_Kewley  \
12353     25035.06        640257.3       4968.048  …           QSO

       class_BPT       kde  class_Kewley_si  class_Kewley_oi    y_WHAN  \
12353        AGN  0.002225              QSO              QSO  100.2064

        z_WHAN group_WHAN subclass_WHAN class_WHAN
12353  78.2348        ELG           SFG        SFG

[1 rows x 28 columns]
```

[140]:
```
print(f"SFG in BPT: {gal[(gal.class_BPT =='SFG')].count()[0]/len(gal) * 100} %"+
      f"\nSFG in WHAN: {gal[(gal.class_WHAN=='SFG')].count()[0]/len(gal) * 100}␣
 ↪%")
```

```
SFG in BPT: 78.56 %
SFG in WHAN: 69.69999999999999 %

C:\Users\Admin\AppData\Local\Temp\ipykernel_10780\3561567233.py:1:
FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a
future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  print(f"SFG in BPT: {gal[(gal.class_BPT =='SFG')].count()[0]/len(gal) * 100}
%"+
C:\Users\Admin\AppData\Local\Temp\ipykernel_10780\3561567233.py:2:
FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a
future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  f"\nSFG in WHAN: {gal[(gal.class_WHAN=='SFG')].count()[0]/len(gal) * 100} %")
```

[141]:
```
print(f"Composite in BPT, SFG in WHAN: {gal[(gal.class_BPT =='composite') &␣
 ↪(gal.class_WHAN=='SFG')].count()[0]/len(gal) * 100} %"+
      f"\nAGN in BPT, SFG in WHAN: {gal[(gal.class_BPT =='AGN') & (gal.
 ↪class_WHAN=='SFG')].count()[0]/len(gal) * 100} %")
```

```
Composite in BPT, SFG in WHAN: 0.65 %
AGN in BPT, SFG in WHAN: 0.13333333333333333 %

C:\Users\Admin\AppData\Local\Temp\ipykernel_10780\2903933832.py:1:
FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a
future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  print(f"Composite in BPT, SFG in WHAN: {gal[(gal.class_BPT =='composite') &
(gal.class_WHAN=='SFG')].count()[0]/len(gal) * 100} %"+
C:\Users\Admin\AppData\Local\Temp\ipykernel_10780\2903933832.py:2:
FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a
```

```
future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  f"\nAGN in BPT, SFG in WHAN: {gal[(gal.class_BPT =='AGN') &
(gal.class_WHAN=='SFG')].count()[0]/len(gal) * 100} %")
```

[142]:
```
print(f"AGN in BPT: {gal[(gal.class_BPT =='AGN')].count()[0]/len(gal) * 100} %"+
      f"\nAGN in WHAN: {gal[(gal.class_WHAN=='AGN')|(gal.class_WHAN=='LINER')].
 ↪count()[0]/len(gal) * 100} %")
```

```
AGN in BPT: 9.406666666666668 %
AGN in WHAN: 30.270000000000003 %

C:\Users\Admin\AppData\Local\Temp\ipykernel_10780\1289124998.py:1:
FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a
future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  print(f"AGN in BPT: {gal[(gal.class_BPT =='AGN')].count()[0]/len(gal) * 100}
%"+
C:\Users\Admin\AppData\Local\Temp\ipykernel_10780\1289124998.py:2:
FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a
future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  f"\nAGN in WHAN:
{gal[(gal.class_WHAN=='AGN')|(gal.class_WHAN=='LINER')].count()[0]/len(gal) *
100} %")
```

[143]:
```
print(f"AGN in BPT: {gal[(gal.class_BPT =='AGN')].count()[0]/len(gal) * 100} %"+
      f"\nStrong AGN in WHAN: {gal[(gal.class_WHAN=='AGN')].count()[0]/len(gal)␣
 ↪* 100} %")
```

```
AGN in BPT: 9.406666666666668 %
Strong AGN in WHAN: 27.636666666666663 %

C:\Users\Admin\AppData\Local\Temp\ipykernel_10780\1692807536.py:1:
FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a
future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  print(f"AGN in BPT: {gal[(gal.class_BPT =='AGN')].count()[0]/len(gal) * 100}
%"+
C:\Users\Admin\AppData\Local\Temp\ipykernel_10780\1692807536.py:2:
FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a
future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  f"\nStrong AGN in WHAN: {gal[(gal.class_WHAN=='AGN')].count()[0]/len(gal) *
100} %")
```

[144]:
```
print(f"AGN in BPT: {gal[(gal.class_BPT =='AGN')].count()[0]/len(gal) * 100} %"+
      f"\nLINER (wAGN & RG) in WHAN: {gal[(gal.class_WHAN=='LINER')].count()[0]/
 ↪len(gal) * 100} %")
```

```
AGN in BPT: 9.406666666666668 %
LINER (wAGN & RG) in WHAN: 2.6333333333333333 %

C:\Users\Admin\AppData\Local\Temp\ipykernel_10780\3831180353.py:1:
FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a
future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  print(f"AGN in BPT: {gal[(gal.class_BPT =='AGN')].count()[0]/len(gal) * 100}
%"+
C:\Users\Admin\AppData\Local\Temp\ipykernel_10780\3831180353.py:2:
FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a
future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  f"\nLINER (wAGN & RG) in WHAN:
{gal[(gal.class_WHAN=='LINER')].count()[0]/len(gal) * 100} %")
```

[145]:
```
print(f"AGN in BPT and WHAN: {gal[(gal.class_BPT =='AGN') & (gal.class_WHAN␣
↪=='AGN')].count()[0]/len(gal) * 100} %"+
      f"\nAGN in BPT but not WHAN: {gal[(gal.class_BPT =='AGN') & (gal.
↪class_WHAN !='AGN')].count()[0]/len(gal) * 100} %"+
      f"\nAGN in WHAN but not BPT: {gal[(gal.class_BPT !='AGN') & (gal.
↪class_WHAN =='AGN')].count()[0]/len(gal) * 100} %")
```

```
AGN in BPT and WHAN: 7.256666666666667 %
AGN in BPT but not WHAN: 2.15 %
AGN in WHAN but not BPT: 20.380000000000003 %

C:\Users\Admin\AppData\Local\Temp\ipykernel_10780\406922451.py:1: FutureWarning:
Series.__getitem__ treating keys as positions is deprecated. In a future
version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  print(f"AGN in BPT and WHAN: {gal[(gal.class_BPT =='AGN') & (gal.class_WHAN
=='AGN')].count()[0]/len(gal) * 100} %"+
C:\Users\Admin\AppData\Local\Temp\ipykernel_10780\406922451.py:2: FutureWarning:
Series.__getitem__ treating keys as positions is deprecated. In a future
version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  f"\nAGN in BPT but not WHAN: {gal[(gal.class_BPT =='AGN') & (gal.class_WHAN
!='AGN')].count()[0]/len(gal) * 100} %"+
C:\Users\Admin\AppData\Local\Temp\ipykernel_10780\406922451.py:3: FutureWarning:
Series.__getitem__ treating keys as positions is deprecated. In a future
version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  f"\nAGN in WHAN but not BPT: {gal[(gal.class_BPT !='AGN') & (gal.class_WHAN
=='AGN')].count()[0]/len(gal) * 100} %")
```

Now you've completed the Python-based data analysis journey within the confines of this book, the exploration of astronomy through GitHub opens up a vast realm of possibilities. GitHub serves as a treasure trove of resources, offering an infinite array of materials across various branches of

astronomy.

Correction Window : astrodingra@gmail.com

Material Link : https://astrodingra.github.io/academics/material.html