Python for astronomy: an overview of astronomical data analysis and database querying packages

R. Virinchi I20PH008
NIT Surat

May 2022

1 Abstract

In this mini project the goal is to become familiar with python along with some widely used packages like numpy, matplotlib and a few packages available specifically for astronomical data analysis like astropy, astroquery, specutils, photutils. We will also take a look at a survey aimed at mapping the night sky, Sloan Digital Sky Survey. We will also explore and list various ways or methods used for downloading FITS files from SDSS archives using astroquery, reading them using astropy and writing them to disk. This will also cover very basic SQL (Structured Query Language) that is needed to use query_sql function of astroquery.

2 SDSS

The Sloan Digital Sky Survey (SDSS) stands as a monumental achievement in observational astronomy, harnessing cutting-edge technology to survey and map celestial objects on an unprecedented scale. This brief review highlights the key technical details that make SDSS a cornerstone in modern astrophysical research.

1. Telescope and Imaging System: SDSS employs a dedicated 2.5-meter wide-field telescope located at Apache Point Observatory in New Mexico, USA. Equipped with a large-format camera, the telescope captures high-quality images in five broadband filters (u, g, r, i, z), spanning ultraviolet to near-infrared wavelengths. This multi-filter approach enables comprehensive photometric measurements and color information for astronomical objects.

2. Photometric Sky Survey: One of SDSS's primary objectives is to conduct a thorough and systematic photometric survey of the night sky. The imaging system captures data with unprecedented precision, detecting a

vast array of celestial objects, including stars, galaxies, and quasars. This extensive survey has provided a wealth of data for large-scale structure studies and cosmological investigations.

3. Spectroscopic Surveys: SDSS incorporates a sophisticated spectrograph capable of obtaining spectra for more than 600 celestial objects simultaneously. This innovative technology allows researchers to analyze the chemical composition, redshift, and other critical properties of galaxies, stars, and quasars. The resulting spectroscopic surveys, such as SDSS-I/II and SDSS-III, have significantly advanced our understanding of the cosmos.

4. Data Management and Processing: Handling the immense volume of data generated by SDSS requires robust data management and processing systems. The project has developed sophisticated pipelines and algorithms to calibrate and reduce the raw data, ensuring high-quality and reliable results. The processed data is then made publicly accessible through dedicated databases, fostering collaboration and enabling a wide range of astronomical studies.

5. SkyServer and Data Access: SDSS's SkyServer platform provides a user-friendly interface for accessing and exploring the vast dataset. With SQL-based queries, astronomers can extract specific information about celestial objects, facilitating customized analyses. The open-access philosophy of SDSS has empowered researchers globally to leverage its wealth of data for various scientific investigations.

6. Ongoing Legacy Surveys: SDSS has evolved over multiple phases, each building on the success of its predecessors. The ongoing SDSS-IV collaboration, comprising the Apache Point Observatory Galactic Evolution Experiment (APOGEE), Mapping Nearby Galaxies at Apache Point Observatory (MaNGA), and Extended Baryon Oscillation Spectroscopic Survey (eBOSS), continues to push the boundaries of observational astronomy, exploring new realms of the universe.

The Sloan Digital Sky Survey stands as a testament to the marriage of advanced technology and ambitious astronomical goals. From its state-of-the-art telescope and imaging system to the innovative spectroscopic surveys and robust data management infrastructure, SDSS has become a cornerstone for astronomers worldwide. Its ongoing legacy surveys promise to unravel even more mysteries of the universe, solidifying SDSS's role as a technical marvel in observational astronomy.

3 FITS

In the realm of astronomical data storage, the Flexible Image Transport System (FITS) file format has established itself as a standard for encoding, archiving, and exchanging scientific data. This review explores the significance of FITS files in astronomy, shedding light on their structural intricacies and the key features that make them indispensable in the field.

1. Data Storage and Portability: FITS files excel at storing a wide range of astronomical data, including images, spectra, and tables. Their self-contained structure allows for the storage of both primary data and metadata in a single file, ensuring portability and ease of sharing among researchers. This characteristic makes FITS an ideal choice for collaborative projects and data dissemination within the astronomical community.

2. Header-Data Unit (HDU) Architecture: A fundamental aspect of FITS files is the Header-Data Unit (HDU) architecture. Each FITS file comprises one or more HDUs, with each HDU consisting of a header and associated data. The header contains metadata, providing essential information about the data structure, telescope parameters, and observational details. The data section holds the actual numerical values representing the astronomical observations.

3. ASCII and Binary Table Extensions: FITS supports both ASCII and binary table extensions, facilitating the storage of tabular data. This versatility is crucial for encoding complex datasets, such as catalogs of celestial objects or multi-dimensional arrays, in a standardized format. Astronomers can seamlessly embed tables within FITS files, enabling efficient organization and retrieval of information.

4. World Coordinate System (WCS):The FITS standard includes provisions for encoding World Coordinate System (WCS) information in the header. This feature is paramount for mapping pixel coordinates to celestial coordinates, allowing astronomers to spatially locate objects within the observed field. WCS parameters in the header enable precise astrometric calibration, a critical aspect of accurate astronomical data analysis.

5. Extensibility and Customization: FITS files offer extensibility, allowing scientists to include custom header keywords and data structures tailored to specific observational requirements. This flexibility ensures that FITS remains adaptable to evolving research needs and diverse astronomical instruments. Customization capabilities make FITS a robust and future-proof choice for data storage.

6. Community Support and Software Ecosystem: The widespread adoption of FITS within the astronomical community has led to the development of a rich software ecosystem. Numerous tools and libraries, such as Astropy in Python, facilitate the creation, manipulation, and analysis of FITS files. This collaborative approach enhances the usability of FITS across different platforms and computational environments.

The FITS file format has stood the test of time as a reliable and versatile standard in astronomical data storage. Its well-defined HDU architecture, support for tables, WCS information, extensibility, and widespread community support make FITS files an indispensable tool for astronomers. As the field continues to evolve, FITS remains a cornerstone, ensuring the efficient exchange and preservation of valuable astronomical data.

4 Python

Python has emerged as a leading programming language in the field of astronomy, revolutionizing the way researchers analyze, visualize, and interpret astronomical data. Its versatility, readability, and extensive library support have made Python an indispensable tool for astronomers worldwide. This section explores how and why Python has become the language of choice in astronomy.

1. Data Handling and Analysis: Python's simplicity and readability make it an ideal language for handling and analyzing astronomical data. The NumPy library provides efficient support for array operations, enabling astronomers to manipulate large datasets with ease. Additionally, the SciPy library offers a comprehensive set of tools for scientific computing, including optimization, integration, and signal processing, which are crucial for processing astronomical data.

2. Visualization: Astronomy heavily relies on data visualization to make sense of complex celestial phenomena. Python excels in this regard, with the Matplotlib library providing a versatile platform for creating publication-quality plots and visualizations. Seaborn and Plotly further enhance the aesthetics and interactivity of visualizations, facilitating the communication of scientific findings to a broader audience.

3. Machine Learning Applications: In recent years, machine learning has gained traction in astronomy for tasks such as classification, regression, and data mining. Python's scikit-learn and TensorFlow libraries are widely adopted for implementing machine learning algorithms. Researchers leverage these tools to automate the identification of astronomical objects, classify galaxies, and predict celestial events, thereby accelerating the pace of astronomical discovery.

4. Astrophysical Simulations: Python is increasingly employed in the simulation of astrophysical processes and the modeling of complex celestial systems. The Astropy library provides a rich set of functionalities for astronomy-specific computations, including unit conversions, coordinate transformations, and time handling. Simulation frameworks such as AstroML and yt enable astronomers to simulate and visualize astrophysical phenomena with remarkable precision.

5. Community Collaboration and Open Source: Python's open-source nature fosters collaboration and information sharing within the astronomy

community. The Astropy Project, a community-driven effort, has developed a core package that standardizes and streamlines common astronomy tasks. Additionally, Jupyter notebooks facilitate reproducibility, allowing researchers to share code, data, and analyses seamlessly.

6. Integration with Observational Instruments: Python's adaptability extends to its integration with observational instruments and telescopes. Astropy provides tools for reading and writing common data formats used in astronomy, ensuring compatibility with various observatories and telescopes. This enables astronomers to process and analyze data directly from these instruments, streamlining the research workflow.

Python's widespread adoption in astronomy is a testament to its versatility, efficiency, and vibrant ecosystem. The language has transformed the way astronomers approach data analysis, visualization, and simulation, accelerating the pace of discovery in the cosmos. As Python continues to evolve, its role in shaping the future of astronomical research is bound to expand, solidifying its place as an indispensable tool for astronomers worldwide.

5 Astropy

Astropy, an open-source astronomy library for Python, has become an integral part of the astronomical community, providing a diverse set of tools and functionalities to streamline data analysis, manipulation, and visualization. This review explores the key aspects of Astropy and highlights some of its most widely used functions that have significantly contributed to advancing astronomical research.

1. Unit and Quantity Handling: Astropy introduces a powerful system for handling physical quantities and units, allowing astronomers to work seamlessly with diverse datasets. The `Quantity` class enables the easy manipulation of numerical values with associated units, promoting consistency and accuracy in calculations.

2. Celestial Coordinate Transformations (Astropy Coordinates): Astropy's `coordinates` module provides extensive support for handling celestial coordinates. The library allows astronomers to transform coordinates between different systems, convert between coordinate frames, and perform celestial geometry calculations. This functionality is crucial for accurate astrometry and positional analysis.

3. Time and Dates: Astropy simplifies the handling of time and dates in astronomical data with its `time` module. Astronomers can convert between different time scales, calculate time intervals, and work with precise time representations, essential for synchronizing observations and coordinating astronomical events.

4. FITS File Handling: Astropy includes functionality for reading, writing, and manipulating FITS files, a standard format in astronomy. The `fits` module allows astronomers to access and modify header information, extract data, and perform various operations on FITS files seamlessly.

5. Cosmological Calculations: Astropy's `cosmology` module provides tools for performing cosmological calculations, including distances, ages, and redshift-related quantities. Astronomers can model and analyze the large-scale structure of the universe, making it easier to interpret observational data in a cosmological context.

6. Statistical Tools: Astropy's `stats` module offers a range of statistical tools for astronomers. From basic descriptive statistics to more advanced statistical analyses, this functionality supports researchers in extracting meaningful insights from observational data and assessing the significance of their findings.

7. Visualization with Matplotlib Integration: Astropy seamlessly integrates with Matplotlib, a widely-used plotting library in Python. Astronomers can create publication-quality plots and visualizations using the `pyplot` module, which simplifies the process of representing complex astronomical data in a clear and informative manner.

8. Physical Constants and Constants Database: Astropy provides a comprehensive set of physical constants through its `constants` module. This feature simplifies calculations involving physical quantities by ensuring access to accurate and up-to-date values.

9. Photometric Systems and Conversions: The `photometry` module in Astropy supports the manipulation of photometric data, including the conversion between different photometric systems. This functionality is essential for standardizing and comparing observations made with diverse instruments and filters.

10. Astronomical Tables and Data Structures: Astropy includes tools for working with astronomical tables and data structures. The `Table` class allows astronomers to manipulate and analyze tabular data efficiently, supporting the organization and storage of diverse datasets.

Astropy stands as a cornerstone in modern astronomy, offering a rich set of functionalities that cater to the diverse needs of researchers. Its seamless integration with the Python ecosystem, coupled with extensive documentation and community support, makes it an invaluable resource for astronomers aiming to extract meaningful insights from their data. As Astropy continues to evolve, its comprehensive set of tools ensures that it will remain a vital companion for astronomers exploring the vast mysteries of the cosmos.

6 Astroquery

Astroquery, a powerful Python library, has emerged as a go-to tool for astronomers seeking seamless access to a plethora of astronomical data archives. This review delves into the significance of Astroquery and highlights several of its key functions, showcasing how it facilitates efficient querying and retrieval of data from various astronomical databases.

1. Unified Query Interface: Astroquery provides a unified and consistent query interface for interacting with multiple astronomical data archives. This abstraction simplifies the querying process, allowing astronomers to seamlessly retrieve data from diverse sources without the need to learn different query languages or protocols.

2. Simplicity in Querying Astronomical Databases: One of the standout features of Astroquery is its simplicity in querying. With a few lines of code, astronomers can construct queries to access data from repositories such as NASA's Astrophysics Data System (ADS), SIMBAD, and more. This streamlined approach enhances the accessibility of astronomical data for researchers.

3. Access to Remote Data: Astroquery enables astronomers to access remote datasets hosted on various astronomical servers. Whether retrieving images, spectra, or catalogs, Astroquery provides a standardized method for fetching data, making it an invaluable tool for observational astronomers and those involved in large-scale surveys.

4. Support for Major Astronomical Archives: Astroquery supports a wide range of major astronomical archives, including but not limited to NASA archives (e.g., MAST, IRSA), European archives (e.g., ESO Archive), and other international repositories. This broad coverage ensures that researchers can tap into a wealth of data resources, fostering collaboration and enabling a diverse range of studies.

5. Coordination with Astropy: Astroquery seamlessly integrates with Astropy, the popular astronomy library in Python. This integration enhances the overall functionality of both libraries, allowing astronomers to combine the querying capabilities of Astroquery with the data analysis and manipulation tools provided by Astropy.

6. Asynchronous Queries: To optimize efficiency, Astroquery supports asynchronous queries. This feature is particularly useful when dealing with large datasets or when making multiple queries concurrently. Asynchronous queries enhance the overall performance and responsiveness of data retrieval processes.

7. Query Caching for Improved Performance: Astroquery incorporates a caching mechanism that stores previously retrieved data locally. This

minimizes redundant queries and significantly improves the performance of subsequent data retrieval tasks. The cache management system is customizable, allowing users to control cache duration and storage.

8. Extensibility and Community Contributions: Astroquery is designed to be extensible, encouraging community contributions and the addition of new modules for interacting with emerging astronomical data archives. The active community surrounding Astroquery ensures that it stays current and continues to adapt to the evolving landscape of astronomical data repositories.

9. Ease of Authentication: For archives that require user authentication, Astroquery simplifies the authentication process. Users can securely manage their credentials, ensuring a smooth and secure interaction with authenticated data sources.

10. Documentation and User Support: Astroquery boasts comprehensive documentation that guides users through the setup process, query construction, and troubleshooting. The availability of documentation and user support forums contributes to the user-friendly nature of Astroquery, making it accessible to astronomers with varying levels of expertise.

Astroquery stands as a pivotal tool in the modern astronomer's toolkit, providing a streamlined and consistent interface for accessing astronomical data archives. Its user-friendly design, integration with Astropy, and support for a wide range of data sources make it an indispensable resource for researchers seeking to explore and analyze the vast cosmic datasets available in the digital age. The continuous development and community engagement ensure that Astroquery remains a dynamic and reliable solution for astronomers navigating the ever-expanding realm of astronomical data.

7 Photutils

Photutils, a Python package tailored for astronomical photometry, has become an essential tool for researchers and astronomers aiming to extract meaningful information from astronomical images. This brief review sheds light on the significance of Photutils and highlights some of its key functions that contribute to accurate and efficient photometric analysis.

1. Aperture Photometry: Photutils excels in aperture photometry, providing a robust framework for measuring flux within specified apertures on astronomical images. The aperture_photometry function allows users to define circular, elliptical, or custom-shaped apertures, facilitating precise extraction of photometric information from diverse sources such as stars or galaxies.

2. Background Subtraction: Accurate photometry often requires proper background subtraction to account for ambient light or instrumental noise. Photutils offers the Background class, which enables users to model and subtract background contributions from images. This feature is crucial for enhancing the accuracy of photometric measurements, especially in crowded or complex regions.

3. PSF Photometry: Photutils supports Point Spread Function (PSF) photometry through the DAOPhotPSFPhotometry class. This functionality allows astronomers to model and fit the PSF of point sources in an image, enabling precise measurements even in regions with overlapping or blended sources.

4. Source Detection and Segmentation: The detect_sources function in Photutils facilitates automatic source detection within images, providing a foundation for subsequent photometric analysis. The associated segmentation module allows users to label and extract distinct sources, streamlining the identification and analysis of multiple objects in crowded fields.

5. Centroiding and Morphological Measurements: Precise centroiding is crucial for accurate photometry. Photutils provides the centroid_sources function for computing centroids of detected sources. Additionally, the morphology module enables users to extract morphological properties, such as the ellipticity and orientation, offering insights into the shapes of astronomical objects.

6. Photometric Aperture Corrections: To account for variations in instrumental and atmospheric conditions, Photutils supports photometric aperture corrections. The aperture_correction function allows users to estimate and apply corrections, ensuring that photometric measurements are standardized and comparable across different observations.

7. Interactive Aperture Photometry with Aperture Photometry Viewer (APV): Photutils incorporates the Aperture Photometry Viewer (APV), a graphical user interface (GUI) that enables interactive aperture photometry. APV allows users to define apertures, perform photometry, and visualize results in real-time, enhancing the user experience and facilitating rapid exploration of data.

8. Integration with Astropy and WCS Handling: As part of the Astropy ecosystem, Photutils seamlessly integrates with Astropy, leveraging its capabilities in handling World Coordinate System (WCS) information. This integration ensures accurate spatial transformations and facilitates compatibility with other astronomical tools and libraries.

9. Open Source and Community Support: Being an open-source project, Photutils benefits from a vibrant community of astronomers and developers. Regular updates, bug fixes, and contributions from the community

ensure that Photutils remains a reliable and evolving solution for astronomical photometry.

Photutils stands as a versatile and powerful toolkit for astronomers engaged in photometric analysis. Its comprehensive set of functions, ranging from aperture photometry to PSF fitting and background subtraction, empowers researchers to extract precise measurements from astronomical images. With its user-friendly design, integration with Astropy, and active community support, Photutils continues to play a pivotal role in advancing the accuracy and efficiency of photometric studies in the field of astronomy.

8 Matplotlib

Matplotlib, a versatile and widely-used Python library, has become synonymous with high-quality data visualization. This brief review explores the significance of Matplotlib and highlights some of its key features that make it an indispensable tool for researchers, scientists, and data enthusiasts.

1. Publication-Quality Plots: Matplotlib excels in creating publication-quality plots with a myriad of customization options. Its intuitive syntax allows users to generate a variety of charts, including line plots, scatter plots, bar plots, and more, ensuring that the visual representation of data is both informative and visually appealing.

2. Matplotlib.pyplot Interface: The pyplot interface in Matplotlib provides a MATLAB-like interface for quick and easy plotting. Users can create and customize plots with just a few lines of code, making it accessible for users with varying levels of programming expertise.

3. Extensive Plot Customization: Matplotlib offers extensive customization options, allowing users to tailor the appearance of plots to meet specific needs. Customizations include color mapping, line styles, markers, annotations, and text placement, providing fine-grained control over the visual elements of a plot.

4. Multi-Dimensional Plotting with 3D and Contour Plots: Matplotlib supports 3D plotting and contour plots, enabling the visualization of multi-dimensional datasets. Researchers can explore relationships between variables in three dimensions, making Matplotlib a powerful tool for scientific visualization and analysis.

5. Seamless Integration with Jupyter Notebooks: Matplotlib seamlessly integrates with Jupyter Notebooks, providing an interactive environment for data exploration and analysis. Plots can be embedded directly into notebooks, allowing researchers to visualize data and share insights within a single document.

6. Rich Set of Colormaps: Matplotlib offers a rich set of colormaps for visualizing scalar and multi-dimensional data. The choice of colormaps enhances the interpretability of plots, allowing users to convey information effectively and intuitively.

7. Animated Plots and Interactivity: Matplotlib supports the creation of animated plots, adding a dynamic dimension to data visualization. Interactivity features such as zooming, panning, and adding widgets further enhance the exploration and understanding of complex datasets.

8. Integration with NumPy Arrays: Matplotlib seamlessly integrates with NumPy, a fundamental library for numerical computing in Python. Users can directly plot NumPy arrays, making it convenient to visualize mathematical functions, experimental data, and simulation results.

Matplotlib remains a cornerstone in the Python ecosystem for data visualization. Its versatility, ease of use, and extensive customization options make it a preferred choice for researchers and data scientists across various domains. As Matplotlib continues to evolve, it continues to empower users to visually communicate insights, explore data relationships, and create compelling visual narratives.

9 Numpy

NumPy, the foundational library for numerical computing in Python, has played a pivotal role in enabling a wide range of scientific and mathematical applications. This brief review explores the significance of NumPy and highlights some of its key functionalities that make it an essential tool for researchers, engineers, and data scientists.

1. Efficient N-Dimensional Array Operations: NumPy's core functionality revolves around its array object, numpy.ndarray. This N-dimensional array enables efficient and vectorized operations, making it a powerful tool for handling large datasets and performing array-centric computations.

2. Universal Functions (ufuncs): NumPy provides a rich set of universal functions (ufuncs) that operate element-wise on arrays. These functions are vectorized, leading to faster and more concise code for mathematical operations on arrays. Ufuncs enhance the performance of numerical computations in comparison to traditional iterative approaches.

3. Broadcasting: NumPy's broadcasting functionality allows for the implicit element-wise operations between arrays of different shapes and sizes. This feature simplifies the syntax for performing operations on arrays with mismatched dimensions, enhancing code readability and reducing the need for explicit looping.

4. Linear Algebra Operations: NumPy includes a comprehensive set of linear algebra operations, making it a valuable tool for solving linear equations, eigenvalue problems, and performing matrix factorizations. These functionalities are crucial for scientific computing, machine learning, and statistical modeling.

5. Random Number Generation: NumPy provides a robust random number generation module, numpy.random, for generating pseudorandom numbers and various probability distributions. This functionality is essential for simulations, statistical analysis, and machine learning applications.

6. Memory Efficiency and Array Broadcasting: NumPy arrays are memory-efficient and provide a homogeneous data structure, allowing for efficient storage and manipulation of large datasets. Additionally, NumPy's broadcasting capabilities enable array operations without unnecessary copying of data, optimizing memory usage.

7. FFT and Signal Processing Tools: NumPy includes a fast Fourier transform (FFT) module for efficient frequency domain analysis. This functionality is valuable in signal processing, image analysis, and various scientific disciplines requiring frequency domain transformations.

NumPy's versatility and efficiency in numerical computing have made it an indispensable tool for a wide range of scientific and engineering applications. Its array-centric approach, extensive functionality, and seamless interoperability with other libraries contribute to its enduring popularity among researchers, engineers, and data scientists. As NumPy continues to evolve, it remains a cornerstone for those seeking robust solutions for numerical computation, data manipulation, and scientific analysis.

10 Specutils

Specutils, a specialized Python library designed for astronomical spectroscopy, plays a crucial role in unraveling the mysteries of celestial spectra. This brief review explores the significance of Specutils and highlights some of its key features that make it an indispensable tool for researchers and astronomers delving into the analysis of spectral data.

1. Unified Interface for Spectral Analysis: Specutils provides a unified and user-friendly interface for spectral analysis, allowing astronomers to seamlessly work with spectral datasets of various formats. This unified approach simplifies the exploration and manipulation of spectral data, regardless of the instrument or wavelength range.

2. Representation of Spectra with Spectrum1D: The core data structure in Specutils is the Spectrum1D class, designed to represent one-dimensional spectra. This versatile class encapsulates the essential components of a

spectrum, including flux, uncertainty, and spectral coordinate information, facilitating straightforward handling and analysis.

3. Spectral Resampling and Binning: Specutils includes functionalities for resampling and binning spectra, allowing researchers to manipulate spectral data at different resolutions or wavelengths. This capability is crucial for comparing or combining spectra from different instruments or observational campaigns.

4. Wavelength Calibration and Flux Calibration: Specutils provides tools for wavelength calibration and flux calibration, essential for converting spectral coordinates to physical units and ensuring accurate flux measurements. These functionalities are particularly important when comparing or combining spectra from different sources.

5. Line Fitting and Continuum Normalization: Specutils supports line fitting and continuum normalization, enabling researchers to identify and quantify spectral features. The library includes tools for fitting lines, estimating continua, and removing or normalizing continuum shapes, facilitating the extraction of meaningful information from spectra.

6. Redshift Handling and Velocity Correction: Specutils includes features for handling redshift and correcting for radial velocity, essential when comparing spectra from objects at different distances or velocities. These tools are valuable for studies involving galaxies, quasars, or any astronomical object with redshifted spectra.

7. Spectral Stacking and Combination: Specutils facilitates spectral stacking and combination, allowing researchers to combine multiple spectra into a single representative spectrum. This capability is valuable for extracting composite spectra from datasets containing multiple observations or sources.

8. Interactive Spectral Plotting: The library offers interactive spectral plotting capabilities, enhancing the visualization of spectral data. Researchers can explore and analyze spectra using customizable plots, facilitating the identification of features and trends in the data.

Specutils stands as an invaluable asset for astronomers engaged in spectral analysis, offering a cohesive and user-friendly framework for working with diverse spectral datasets. Its unified interface, extensive functionality, and integration with Astropy make it a pivotal tool for researchers seeking to extract meaningful insights from the rich information encoded in astronomical spectra.

11 Querying the archive

11.1 Getting images

```
get_images(*, coordinates=None, radius=<Quantity 2. arcsec>,
matches=None, run=None, rerun=301, camcol=None, field=None,
band='g', timeout=60, cache=True, get_query_payload=False,
data_release=17, show_progress=True)[source]
```

Download an image from SDSS. Querying SDSS for images
will return the entire plate. For subsequent
analyses of individual objects the query can be made
with one the following groups of parameters
(whichever comes first is used):

-matches (result of a call to query_region);

-coordinates, radius;

-run, rerun, camcol, field.

Parameters:
coordinatesstr or astropy.coordinates object:
The target around which to search. It may be specified as a string in
which case it is resolved using online services or as the appropriate
astropy.coordinates object. ICRS coordinates may also be entered as
strings as specified in the astropy.coordinates module.

radiusstr or Quantity object, optional:
The string must be parsable by Angle. The appropriate Quantity object
from astropy.units may also be used. Defaults to 2 arcsec.

matchesTable:
Result of query_region.

runinteger, optional:
Length of a strip observed in a single continuous image observing scan.

reruninteger, optional:
Reprocessing of an imaging run. Defaults to 301 which is the most recent rerun.

camcolinteger, optional:
Output of one camera column of CCDs.

fieldinteger, optional:
Part of a camcol of size 2048 by 1489 pixels.

bandstr or list:
Could be individual band, or list of bands. Options: 'u', 'g', 'r', 'i', or 'z'.

timeoutfloat, optional:

Time limit (in seconds) for establishing successful connection with remote server. Defaults to SDSSClass.TIMEOUT.

get_query_payloadbool, optional:
If True, this will return the data the query would have sent out, but does not actually do the query.

cachebool, optional:
Cache the images using astropy's caching system

data_releaseint, optional:
The data release of the SDSS to use.

show_progressbool, optional:
If False, do not display download progress.

Returns:
listList of HDUList objects.

11.2 Getting spectra

```
get_spectra(*, coordinates=None, radius=<Quantity 2. arcsec>,
matches=None, plate=None, fiberID=None, mjd=None, timeout=60,
get_query_payload=False, data_release=17, cache=True,
show_progress=True)[source]
```

Download spectrum from SDSS.
The query can be made with one the following groups of parameters (whichever comes first is used):

-matches (result of a call to query_region);

-coordinates, radius;

-plate, mjd, fiberID.

Parameters:
coordinatesstr or astropy.coordinates object:
The target around which to search. It may be specified as a string in which case it is resolved using online services or as the appropriate astropy.coordinates object. ICRS coordinates may also be entered as strings as specified in the astropy.coordinates module.

radiusstr or Quantity object, optional:
The string must be parsable by Angle. The appropriate Quantity object from astropy.units may also be used. Defaults to 2 arcsec.

matchesTable:
Result of query_region.

plateinteger, optional:
Plate number.

fiberIDinteger, optional:
Fiber number.

mjdinteger, optional:
Modified Julian Date indicating the date a given piece of SDSS data was taken.

timeoutfloat, optional:
Time limit (in seconds) for establishing successful connection with remote server.
Defaults to SDSSClass.TIMEOUT.

get_query_payloadbool, optional:
If True, this will return the data the query would have sent out,
but does not actually do the query.

data_releaseint, optional:
The data release of the SDSS to use. With the default server,
this only supports DR8 or later.

cachebool, optional:
Cache the spectra using astropy's caching system

show_progressbool, optional:
If False, do not display download progress.

Returns:
listList of HDUList objects.

11.3 Querying a particular region

query_region(*args, **kwargs)

Queries the service and returns a table object.
Used to query a region around given coordinates. Either radius or
width must be specified.

When called with keyword radius, a radial or \cone" search is
performed, centered on each of the given coordinates. In this mode,
internally, this function is equivalent to the object cross-ID

(query_crossid), with slightly different parameters. Note that in this mode there is a server-side limit of 3 arcmin on radius.

When called with keyword width, and optionally a different height, a rectangular search is performed, centered on each of the given coordinates. In this mode, internally, this function is equivalent to a general SQL query (query_sql). The shape of the rectangle is not corrected for declination (i.e. no
correction); conceptually, this means that the rectangle will become increasingly trapezoidal-shaped at high declination.

In both radial and rectangular modes, this function returns all objects within the search area; this could potentially include duplicate observations of the same object.

Parameters:
coordinatesstr or astropy.coordinates object or (list or Column) of coordinates:
The target(s) around which to search. It may be specified as a string in which case it is resolved using online services or as the appropriate astropy.coordinates object. ICRS coordinates may also be entered as strings as specified in the astropy.coordinates module.

Example: ra =
np.array([220.064728084,220.064728467,220.06473483]) dec =
np.array([0.870131920218,0.87013210119,0.870138329659])
coordinates = SkyCoord(ra, dec, frame='icrs', unit='deg')

radiusstr or Quantity object, optional:
The string must be parsable by Angle. The appropriate Quantity object from astropy.units may also be used. The maximum allowed value is 3 arcmin.

widthstr or Quantity object, optional:
The string must be parsable by Angle. The appropriate Quantity object from astropy.units may also be used.

heightstr or Quantity object, optional:
The string must be parsable by Angle. The appropriate Quantity object from astropy.units may also be used. If not specified, it will be set to the same value as width.

timeoutfloat, optional:
Time limit (in seconds) for establishing successful connection with remote server.

```
Defaults to SDSSClass.TIMEOUT.

fieldslist, optional:
SDSS PhotoObj or SpecObj quantities to return. If None, defaults
to quantities required to find corresponding spectra and images
of matched objects (e.g. plate, fiberID, mjd, etc.).

photoobj_fieldslist, optional:
PhotoObj quantities to return. If photoobj_fields is None and
specobj_fields is None then the value of fields is used

specobj_fieldslist, optional:
SpecObj quantities to return. If photoobj_fields is None and
specobj_fields is None then the value of fields is used

obj_namesstr, or list or Column, optional:
Target names. If given, every coordinate should have a
corresponding name, and it gets repeated in the query result.

spectrobool, optional:
Look for spectroscopic match in addition to photometric match?
If True, objects will only count as a match if photometry and
spectroscopy exist. If False, will look for photometric matches only.

field_help: str or bool, optional:
Field name to check whether a valid PhotoObjAll or SpecObjAll
field name. If True or it is an invalid field name all the valid
field names are returned as a dict.

get_query_payloadbool, optional:
If True, this will return the data the query would have sent
out, but does not actually do the query.

data_releaseint, optional:
The data release of the SDSS to use.

cachebool, optional:
If True use the request caching mechanism.

Returns:
A Table object.
```

11.4 Basic SQL syntax

An SQL query consists of three pieces, or blocks: the select block, the from block and the where block.The select block tells the database which columns of data you want it to return. You must separate each column name with a comma. For example, if you wanted to find the celestial coordinates right ascension (ra) and declination (dec) of an object, the select block might read

```
select
ra, dec
```

The from block specifies which table (or tables) you want to search. If you wanted to retrieve information from the specObj table, the from block would read

```
from
specObj
```

The where block allows you to search for records with certain characteristics. Your list of characteristics must be separated by the word ”AND”. Suppose you wanted to limit your search to a patch of sky with ra between 140 and 141 degrees and dec between 20 and 21 degrees. To search only this patch of sky, your where block would read

```
where
ra BETWEEN 140 AND 141 AND
dec BETWEEN 20 AND 21
```

The database will return only those records that have an ra between 140 and 141 and a dec between 20 and 21.

11.5 Querying using SQL

```
query_sql(*args, **kwargs)
Queries the service and returns a table object.

Parameters:
sql_querystr:
An string of SQL query

timeoutfloat, optional:
Time limit (in seconds) for establishing successful connection
with remote server. Defaults to SDSSClass.TIMEOUT.

data_releaseint, optional:
The data release of the SDSS to use.

cachebool, optional:
If True use the request caching mechanism.

Returns:
table: A Table object.
```

12 Reading/Writing data from/to disk

Querying only give us the information we need but it doesn't write the data into out computers. For that we use fits class from astropy.io package. The HDUtable object does have an inbuilt function to write tables to disk but it is only limited to tables (as apparent from its name) so cannot be used if we want to write the whole FITS file or mutiple tables into a single file.

12.1 Code for Querying and Writing files

```
#import packages
from astroquery.sdss import SDSS
from astropy.table import Table
from astropy.coordinates import SkyCoord
from astropy import units as u

#convert to SkyCoord object
units='deg'
coord=SkyCoord(10, 20, unit=units)

#get meta data about the objects in that area of sky
res_table = SDSS.query_region(coord, radius=0.01*u.deg, spectro=True)
res_table = SDSS.query_sql("SELECT top 10 * from specObj where class='star'")

#write each column of the table as seperate fits file to disk
for i in range(len(res_table)):
    res_table[i:i+1].write(f'path\of\file.fits',overwrite=True)
#tip: term to the right side of colon is not included
```

12.2 Code for reading files

```
from astropy.io import fits

# Read FITS file containing spectrum
fits_file_path = 'D:\\pythonenvi\\src\\spectra\\spec_0'
hdul = fits.open(fits_file_path)
data = hdul[1].data
hdul.close()
```

13 Conclusion

In the realm of astronomy, the adoption of Python and its associated libraries, including Matplotlib, NumPy, and Astropy, has transformed the landscape of data analysis, visualization, and computational research. Together, these tools contribute significantly to the efficiency, versatility, and collaborative nature of modern astronomical investigations.