Efficient data preparation with Python

ANACONDA.

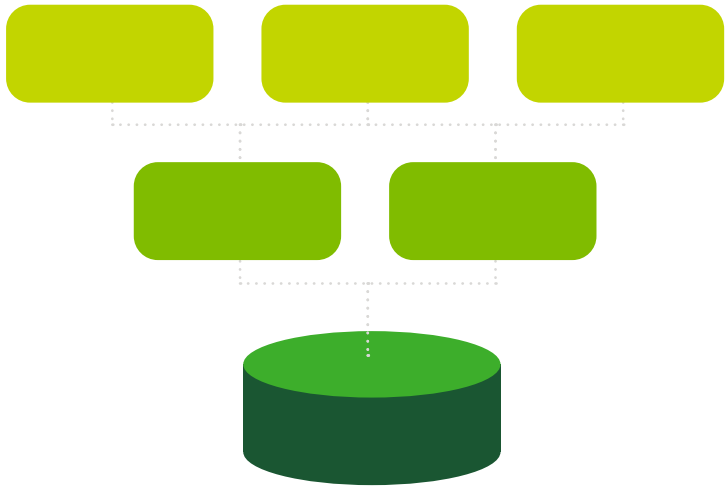
What's Inside

Introduction

Although data scientists spend less time now than they did a few years ago on data preparation, our 2020 State of Data Science survey found that nearly half of their time (45%) is still spent preparing for analysis and modeling.

Data discovery and data preparation have always been among the most time-intensive steps of a research project and for good reason: If there are unaddressed errors in data, or if the wrong version of data is used, there will be errors in the resulting analysis or model. In some cases these errors could render the analysis or model completely useless. This is why thorough data preparation is essential for accurate analyses and accurate models.

Data preparation will likely always be a major step in the data science process. However, data scientists can speed up the time spent on data prep tasks with a well-documented and curated data catalog, a repository of data cleaning functions, and of course, Python tools and libraries created especially for more efficient data prep.



What does data preparation include?

Data preparation is a general term that includes multiple steps that might be needed to prepare data for analysis or machine learning and model development. Data preparation will usually include anonymization, data discovery and exploration, data cleansing, and what some refer to as ETL — extracting, transforming and loading.

Let's take a look at some tools and tips to make each of these steps more efficient.

Data anonymization

Data must be managed ethically and securely throughout the data science process. When working with data that involves personally identifiable information (PII) or other sensitive data, the first step in the data preparation process must be to anonymize the data. One way to do this is to replace real personal data with similar but fake information. An open-source Python tool that helps with this process is **Faker**. Faker's library contains a comprehensive set of data generators for data needed in a variety of domains.

Learn more about Faker and how to install it.

Scrubadub is another Python tool that can be used to efficiently scrub PII from free text, including names, email addresses, phone numbers, social security numbers, usernames, and more.

Read more about Scrubadub and how to easily install this tool.

To maintain the integrity and usefulness of the data after anonymization, the fake data must preserve the structure and semantics of the original data. Consider the relationships that originally existed between fields. For example, how do people's names and employer names relate to the structure of their email addresses? If this is important for the analysis or model, create a mapping of real domains to fake domains. Deduplication should also be performed after anonymization. This is covered in the Data Cleansing section of this guide.

Data discovery and extraction

Finding and accessing the right data sets to answer research questions can be extremely time-consuming tasks without a well-documented, curated data catalog. To make data discovery and extraction more efficient, data engineers should commit to the development of a thorough data ingest and cataloging pipeline and work with IT to ensure data scientists can access data using instructions in the catalog. This labor-intensive prep work will be well worth the time, enabling data scientists to quickly search for and access the data sets they need without involving IT, or worse loading the data only to discover it isn't right for their model. Furthermore, catalogs, when kept up to date, will ensure that data scientists always get the *latest version* of datasets, and perhaps choose between sections of data meant for different uses.

The data catalog should include searchable metadata (type of data, free word text, tags, provenance, etc.) with descriptions of data sets that data scientists would find useful. Each catalog entry should be a definition of how to read a dataset, including the type of data, the loader and arguments required, and the location of files and credentials required to fetch the data. Anaconda recommends using Intake to create data catalogs that empower data scientists to search and load data for their projects easily from Python. Intake allows data scientists to search for data sets in a hierarchical tree, which is more efficient than going through lists of files.

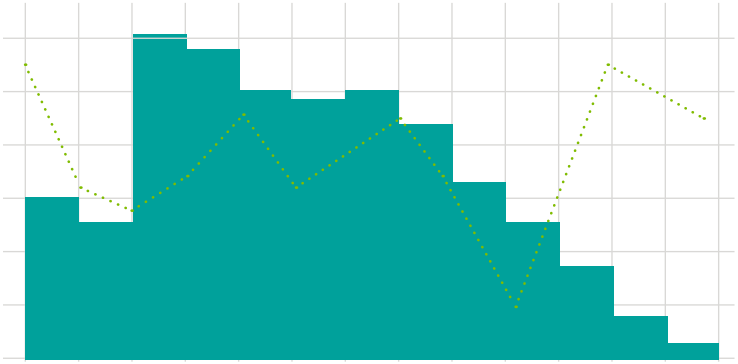
Intake also allows data engineers to provide descriptions of how to load data so data scientists can access data sets without having to figure it out on their own or wait for help from IT. Lastly, Intake allows for the separation of the definition of data sources from their use and analysis. This means data engineers can decide and execute on where data should be stored, in what format, and in what type of package without affecting data scientists' work. The data scientist can then fetch the data in a familiar container (dataframe, data, etc.) that they know how to analyze.

Learn more about Intake and how to get started.

Data exploration and profiling

Once you can access the data, the next step is to conduct an exploratory data analysis (EDA) to gain a better understanding of trends and important characteristics. An EDA allows data scientists to uncover patterns and irregularities in data and to understand frequency counts, variance, and correlations.

For columnar data, the go-to exploration tool is **pandas**, where you can explore descriptive statistics, summarize unique values and missing values, and more. Here we'll provide some additional open-source analysis and visualization tools data scientists can use to explore and understand data sets.



Fast data profiling

pandas_profiling is a Python tool that generates interactive reports from pandas data frames, allowing for quick data profiling with just a few lines of code. **pandas_profiling** reports include:

- More powerful type inference than what Pandas offers
- Essentials such as unique and missing values
- Descriptive stats (mean, mode, standard deviation, quantiles, histograms)
- Most common values
- Correlations
- Missing values matrix, heatmaps, and dendrograms
- Text analysis
- File and image analysis

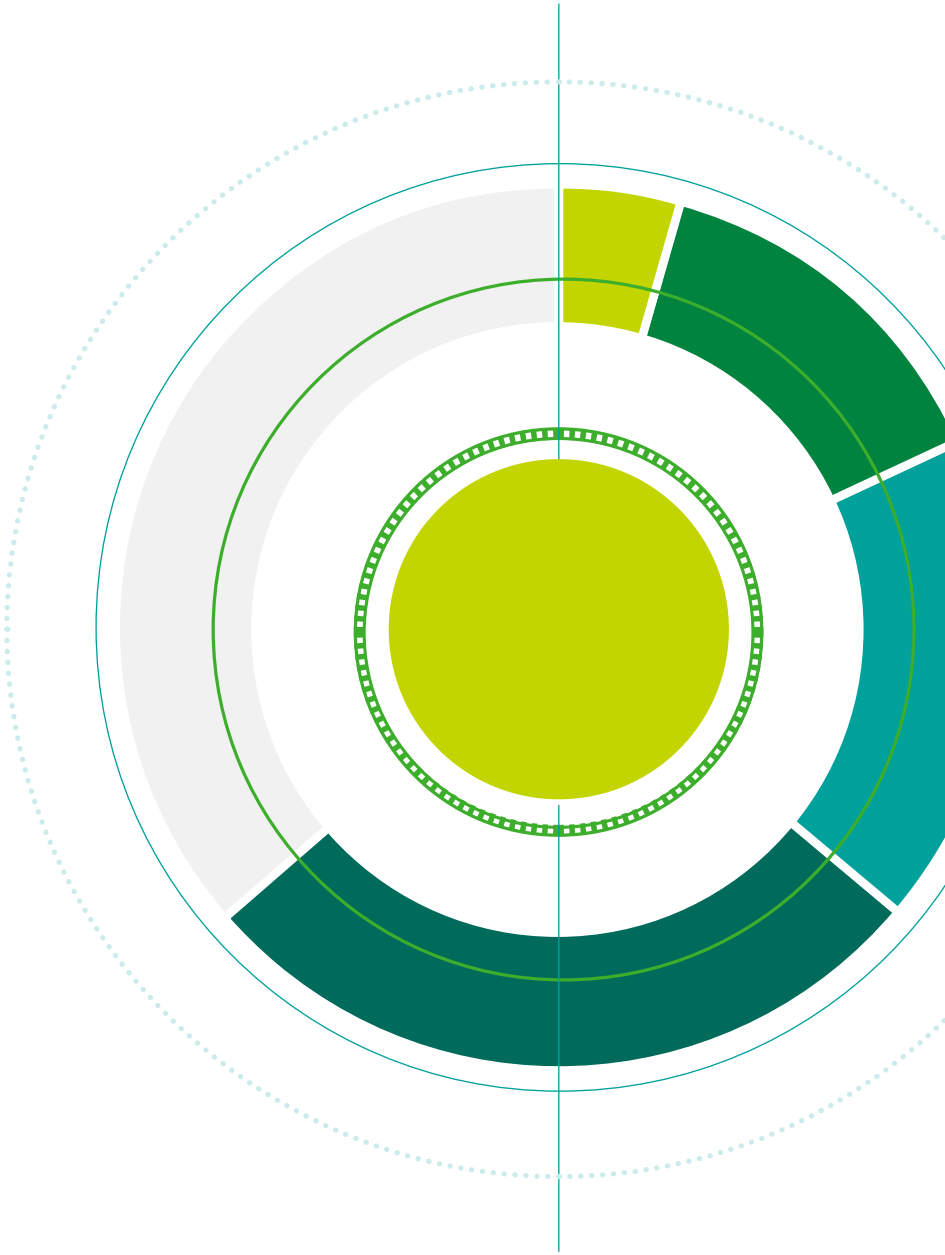
Learn more about pandas_profiling.

Data visualization

Blindly running averages or other statistics on your data can easily lead to incorrect conclusions if there are outliers or trends you did not expect. If you can present your data visually, our brains are highly skilled at spotting patterns and inconsistencies. Thus it is crucial to explore the underlying data visually before drawing conclusions or taking actions. Scatter plots, cluster size analysis, and correlation heat maps can be especially helpful in providing insights when exploring new datasets.

Data exploration is an art, and there is no one perfect way to visualize any data set. There are dozens of "viz" libraries available for Python but here we'll focus on three that are particularly focused on helping you understand the properties of a new dataset: **Seaborn**, **Datashader**, and **Holoviz**.

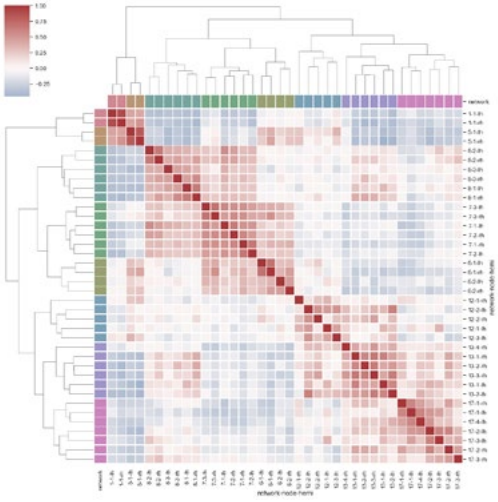
Visit PyViz.org for a complete list of Python visualization libraries.



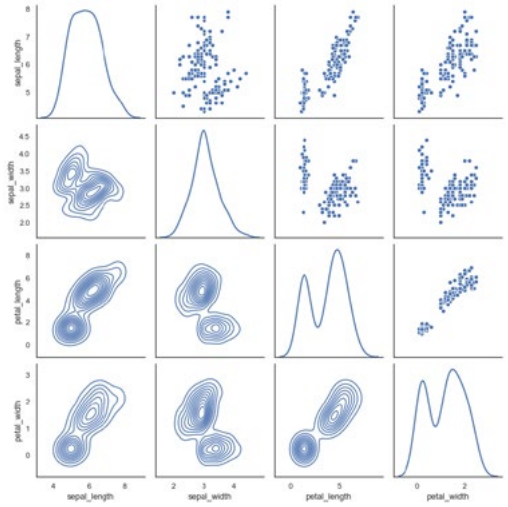
Seaborn

Seaborn is a library built on top of Matplotlib that provides a large variety of visualization tools for data exploration, focusing on statistical visualizations. It is closely integrated with pandas and provides a dataset-oriented API for examining relationships between multiple variables. Here are a few examples of using visualization to explore data sets with Seaborn:

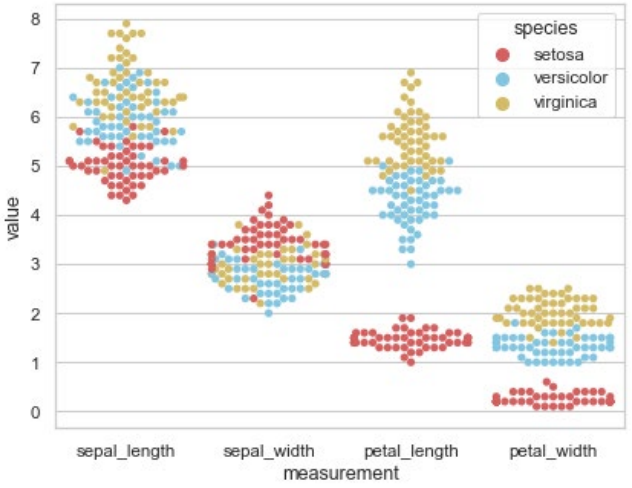
Discovering structure in heatmap data



Paired density and scatterplot matrix



Scatterplot with categorical variables



Datashader

Datashader was created to make it easier to interactively visualize large, multidimensional datasets, with a goal of accurately rendering both the trends and outliers in the data fast enough to be practical for interactive exploration. As illustrated in the figure below, plotting even 1% of a dataset with 300 million points of US Census locations in a standard plotting program (Matplotlib in this case) takes 5 seconds, which is too slow for interactive panning and zooming. Standard plotting tools also require adjusting multiple parameters before they will reveal the data, which is difficult if you don't already know the properties of your dataset! Datashader uses server-side aggregation techniques to accurately reveal the shape of the entire distribution without needing a parameter exploration, in under 0.1 seconds for the entire dataset.

Datashader for painlessly exploring large datasets

Plotting 1% of the data with Matplotlib takes 5 seconds and shows only limited information

Patterns can be revealed by adjusting Matplotlib parameters, if you know what to expect

Datashader renders all 300 million points in 0.1 second with no parameter tuning needed

Holoviz

HoloViz is an Anaconda project that provides tools for high-level and large-data visualization built on top of Matplotlib, Bokeh, and Plotly. HoloViz includes hvPlot for easy one-line plotting from Pandas, Xarray, or Dask to encourage you to visualize every step of your processing pipeline, plus Datashader so that you can render *all* your data without making any assumptions about its content, and Panel for creating web applications for interactive exploration in Jupyter that can later be deployed for sharing with stakeholders. hvPlot works similarly to Seaborn or the default plotting for Pandas, but provides interactive visualizations that support easy exploration.

Data cleansing

Data cleansing or cleaning may be the most plodding task for some data scientists, but it is essential for accurate models and analyses. Data cleansing involves finding and remediating missing, duplicated, irregular, unnecessary and inconsistent data. Some of this dirty data will likely already have been identified in the data exploration phase.

One way to make the data cleansing process more efficient is to maintain a library of functions or a repository of rules that data scientists can refer to when cleaning data sets.

There are several ways to clean data using Python and common open-source libraries such as pandas and NumPy and common visualization tools such as Matplotlib, Seaborn, and HoloViz.

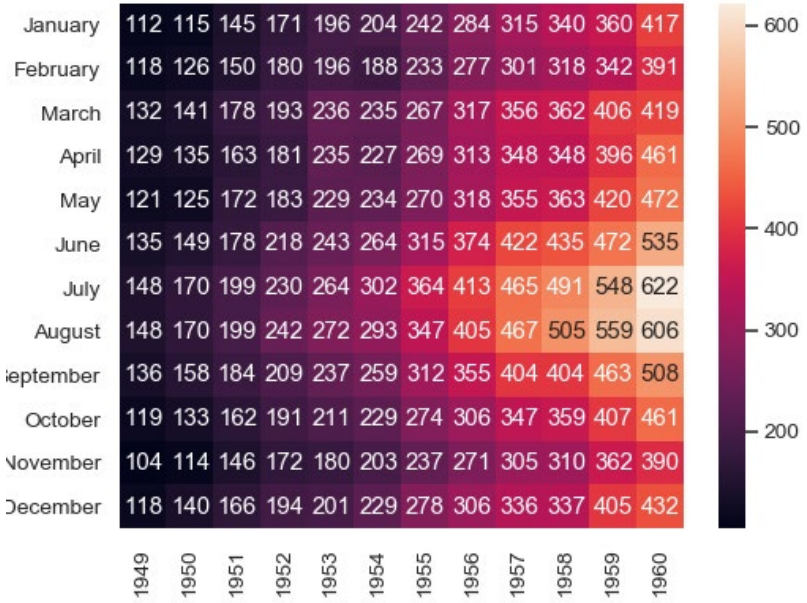
Locating missing data

One way to facilitate the search for missing data is to create a heatmap that shows missing fields in a contrasting color. By looking at a data set this way, a data scientist can quickly discover which features need attention and exclude them altogether if they are not important to the project at hand. Data scientists can use Seaborn and other data visualization tools to quickly create heatmaps with contrasting colors to find missing values or histograms to discover the frequency of missing values.

Alternatively, one can quickly gain an understanding of what data is missing by using Python to create a list of the percentage of missing data for each column:

```
# % of missing
for col in df.columns:
    pct_missing = np.mean(df[col].isnull())
    print('{} - {}%'.format(col, round(pct_missing*100)))
```

Missing Values Heatmap Created with Seaborn



Finding outliers

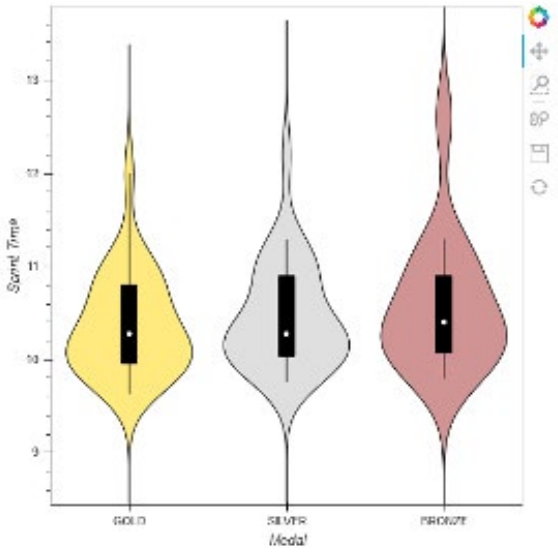
When exploring a data set, it's important to identify outliers and determine if they are in fact outliers or mistakes in data collection. Histograms can also be used to find outliers, as well as bar charts and descriptive statistics. To see a quick description of a data set, input the following:

```
df['feature_name'].describe()
```

This will provide total count, mean, quartiles, and max value, among other characteristics, enabling a quick glance at the distribution between minimum and maximum values.

Visualization tools are also great for identifying outliers, especially scatter plots, clustering, and violin plots. Use Seaborn or HoloViz tools to easily create these visualizations to view data distributions and identify outliers.

Violin Plot Showing Distributions, Created with HoloViz



Identifying duplicates

In addition to removing any features that are unrelated to answer the research questions at hand, duplicates must be removed so that they do not bias the results. Sorting to help identify and then removing duplicates is easy with Python and pandas. Here is an example of sorting by first name (useful if inspecting the data manually) and dropping duplicates:

```
# sorting by first name
data.sort_values("First Name",
inplace=True)

# dropping duplicate values
data.drop_
duplicates(keep=False,inplace=True)
```

Identifying inconsistencies

Common inconsistencies in data sets include capitalization, data formats, and misspelled words. There are a few simple operations in Python for remediating these common inconsistencies. To avoid confusion from variations in capitalization, it's often easiest to make all data lowercase:

```
# make everything lower case.
df['sub_area_lower'] = df['sub_area'].str.lower()
df['sub_area_lower'].value_counts(dropna=False)
```

To identify and remediate misspelled words, use fuzzy logic. To start, search for a few common entries and variations within a category. This example uses cities.

```
from nltk.metrics import edit_distance

df_city_ex = pd.DataFrame(data={'city': ['tokyo', 'tokoyo',
'tokiyo', 'yokohama', 'yokohoma', 'yokahama', 'osaka', 'kobe']})

df_city_ex['city_distance_tokyo'] = df_city_ex['city'].
map(lambda x: edit_distance(x, 'tokyo'))
df_city_ex['city_distance_yokohama'] = df_city_ex['city'].
map(lambda x: edit_distance(x, 'yokohama'))
Df_city_ex
```

This gives us a list of distances (or how many letters are off) from the correct spelling of Tokyo and Yokohama. You will find that misspelled words are typically only a distance of 1 or 2 from the correct word. Greater distances likely indicate a different city. To remediate misspelled words in this category, we can focus on those with a distance of two or less from the correct spelling.

To remediate this:

```
msk = df_city_ex['city_distance_tokyo'] <= 2
df_city_ex.loc[msk, 'city'] = 'tokyo'

msk = df_city_ex['city_distance_yokohama'] <= 2
df_city_ex.loc[msk, 'city'] = 'yokohama'

df_city_ex
```

Obviously, great care must be taken to preserve names which are genuinely different but happen to be similar as separate identifiers.

Another common formatting problem is making address entries uniform. Address entries often contain a combination of spacing, punctuation, and capitalization inconsistencies. To remediate this, we can use a combination of functions to make all values lower case and standardize punctuation.

See detailed examples of how to use Python to remove duplicates, find and correct misspelled words, make capitalization and punctuation uniform, find inconsistencies, make address formatting uniform and more in this detailed data cleaning guide published on Towards Data Science.

Date and time formatting

Data and time format is also often inconsistent. We can use Python to standardize date and time formats, to convert string to date and vice versa, compare and change time zones, and calculate time differences. Here are some useful Python directives to get started:

DIRECTIVE	MEANING	EXAMPLE
%a	Weekday as abbreviated name	Sun, Mon, Tue...
%A	Weekday as full name	Sunday, Monday...
%w	Weekday as a decimal number; 0 is Sunday, 6 is Saturday	0, 1, 2...
%d	Day of month as a zero-padded decimal number	01, 02, 03...
%b	Month as abbreviated name	Jan, Feb, Mar...
%B	Month as full name	January, February...
%m	Month as zero-padded decimal number	01, 02, 03...
%y	Year without century as zero-padded decimal number	98, 99, 00, 01...
%Y	Year with century as a decimal number	1998, 1999, 2000...

Source: Towards Data Science. To learn more about how to use these directives read How to Manipulate Date and Time in Python like a Boss.

Data transformation

Data transformation is the process of modifying or converting data format or structure.
This can include aggregating columns or categories, converting units, converting times and dates.

There are a variety of Python packages available to assist with units conversions. Here are a few of them to note:

- Arrow - a library that simplifies date/time conversion and formatting
- QuantiPhy - a library for converting physical quantities; supports SI scale factors
- Astropy - a library for astronomy that includes comprehensive units/dimensionality handling
- Pint - a Python package to define, operate and manipulate physical quantities
- SymPy - a module that allows for symbolic manipulation of variables, which can represent quantities with units

There are many other Python packages out there to assist with domain-specific conversions.

Automating data prep workflows

Particularly for data that is updated frequently, data engineers and data scientists can speed up the data preparation process by automating some of their workflows. Data professionals can accelerate the data preparation process with these Python-based tools.

Airflow

One way to do this in the open-source world is with Apache Airflow. Airflow is a Python-based data workflow management system originally developed by Airbnb's data team to help speed up their data pipelines. It can be used to schedule tasks to aggregate, cleanse, and organize data, among other tasks unrelated to data preparation.

Learn more about Airflow.

Celery

Celery is a workflow automation tool that was developed with Python, but with a protocol that can be implemented in any language. Celery uses task queues to distribute work across machines or threads and allows for high availability and horizontal scaling. Celery can process millions of tasks per minute and can run on single machines or multiple. A Celery system can even run across data centers.

Learn more about Celery.

Luigi

Luigi is a Python workflow management tool originally developed at Spotify. Luigi is great for managing long-running batch processes, and it's easy to build long-running pipelines that take days or weeks to complete. Luigi's interface is less user-friendly than Airflow's, but it allows for custom scheduling and has a comprehensive library of stock tasks and target data systems.

Learn more about Luigi.

Prefect

Prefect is a workflow management platform that allows users to automate anything they can do with Python. Designed and built with Dask in mind, Prefect schedules workflows while allowing Dask to schedule and manage the resources for tasks within workflows. Prefect generally allows for more flexibility and more dynamic workflows than Airflow or Luigi. For example, Airflow's scheduler is critical to the execution of multiple stages in a workflow, while Prefect decouples this logic into separate processes. Airflow's design makes it better for managing larger tasks while Prefect is better for smaller, modular tasks.

Learn more about Prefect.

Tools are no substitute for human intuition

While there are tools that make the data preparation process more efficient, and automation tools will become more sophisticated over time, it's not easy to use them to apply a critical lens to data. Human experience and intuition are necessary and will continue to be necessary to evaluate data sets and determine if outliers and inconsistencies are errors or simply unusual given the context. Human intuition is also needed to determine whether a data set can contribute valuable insight to solve specific problems or provide relevant answers to the research questions at hand. Machines will not take over the task of data preparation any time soon.

About Anaconda

With more than 20 million users, Anaconda is the world's most popular data science platform and the foundation of modern machine learning. We pioneered the use of Python for data science, champion its vibrant community, and continue to steward open-source projects that make tomorrow's innovations possible. Our enterprise-grade solutions enable corporate, research, and academic institutions around the world to harness the power of open-source for competitive advantage, groundbreaking research, and a better world.

Visit https://www.anaconda.com **to learn more.**

ANACONDA.