While it is challenging to precisely determine the exact 20% of Python libraries that are used 80% of the time, there are certain libraries that are widely used and considered fundamental in various Python projects. Here is a list of some popular Python libraries that are commonly used:

**NumPy**: A library for efficient numerical computations and array operations.

**Pandas**: A data manipulation and analysis library that provides data structures like DataFrames.

**Matplotlib**: A plotting library for creating visualizations and graphs.

**Scikit-learn**: A machine learning library that provides tools for classification, regression, clustering, and more.

**TensorFlow**: A popular deep learning library for building and training neural networks.

**PyTorch**: A deep learning library known for its dynamic neural network architecture.

**Requests**: A library for making HTTP requests and working with APIs.

**Beautiful Soup**: A library for web scraping and parsing HTML/XML documents.

**Django**: A high-level web framework for building web applications.

**Flask**: A lightweight web framework for developing web applications.

**SQLAlchemy**: A SQL toolkit and Object-Relational Mapping (ORM) library for working with databases.

**pytest**: A testing framework for writing and running tests in Python.

**Celery**: A distributed task queue library for handling asynchronous tasks.

**SciPy**: A library for scientific and technical computing, providing modules for optimization, linear algebra, integration, etc.

**OpenCV**: A computer vision library that offers a wide range of image and video processing functions.

**NLTK**: The Natural Language Toolkit provides tools and resources for working with human language data.

**Pygame**: A cross-platform library for game development.

**Pillow**: A library for image processing and manipulation.

**Flask-RESTful**: An extension for Flask that simplifies building RESTful APIs.

**Gunicorn**: A WSGI HTTP server for deploying Python web applications.

Please note that this is not an exhaustive list, and the choice of libraries depends on the specific requirements of your projects. Different domains and use cases may require additional specialized libraries.

**Numpy**

|  |  |  |
| --- | --- | --- |
| **Method** | **Description** | **Example** |
| np.ones() | Creates an array filled with ones. | ones\_arr = np.ones((3, 4)) - Creates a 2-dimensional array filled with ones of shape (3, 4). |
| np.zeros() | Creates an array filled with zeros. | zeros\_arr = np.zeros((2, 3)) - Creates a 2-dimensional array filled with zeros of shape (2, 3). |
| np.arange() | Returns evenly spaced values within a given interval. | arr = np.arange(0, 10, 2) - Creates a 1-dimensional array with values [0, 2, 4, 6, 8]. |
| np.linspace() | Returns evenly spaced numbers over a specified range. | linspace\_arr = np.linspace(0, 1, num=5) - Creates a 1-dimensional array with 5 equally spaced values between 0 and 1. |
| np.reshape() | Reshapes an array into a specified shape. | reshaped\_arr = np.reshape(arr, (2, 3)) - Reshapes the array arr into a 2-dimensional array with shape (2, 3). |
| np.transpose() | Transposes the dimensions of an array. | transposed\_arr = np.transpose(arr) - Transposes the dimensions of the array arr. |
| np.sum() | Computes the sum of array elements. | sum\_val = np.sum(arr) - Computes the sum of all elements in the array arr. |
| np.mean() | Computes the mean of array elements. | mean\_val = np.mean(arr) - Computes the mean value of all elements in the array arr. |
| np.max() | Returns the maximum value in an array. | max\_val = np.max(arr) - Returns the maximum value in the array arr. |
| np.min() | Returns the minimum value in an array. | min\_val = np.min(arr) - Returns the minimum value in the array arr. |
| np.dot() | Computes the dot product of two arrays. | dot\_product = np.dot(arr1, arr2) - Computes the dot product of arrays arr1 and arr2. |
| np.concatenate() | Joins arrays along a specified axis. | concat\_arr = np.concatenate((arr1, arr2), axis=0) - Concatenates arrays arr1 and arr2 along the 0th axis. |
| np.split() | Splits an array into multiple sub-arrays. | sub\_arrays = np.split(arr, 3) - Splits the array arr into 3 sub-arrays. |
| np.argmax() | Returns the indices of the maximum values in an array. | max\_indices = np.argmax(arr) - Returns the indices of the maximum values in the array arr. |
| np.unique() | Returns unique elements in an array. | unique\_vals = np.unique(arr) - Returns the unique elements in the array arr. |
| np.random.rand() | Generates random numbers from a uniform distribution. | rand\_nums = np.random.rand(5) - Generates an array of 5 random numbers between 0 and 1. |
| np.random.randint() | Generates random integers within a specified range. | rand\_ints = np.random.randint(1, 10, size=(3, 3)) - Generates a 2-dimensional array of random integers between 1 and 10 of shape (3, 3). |
| np.exp() | Computes the exponential of array elements. | exp\_vals = np.exp(arr) - Computes the exponential of all elements in the array arr. |
| np.log() | Computes the natural logarithm of array elements. | log\_vals = np.log(arr) - Computes the natural logarithm of all elements in the array arr. |
| np.sin() | Computes the sine of array elements. | sin\_vals = np.sin(arr) - Computes the sine of all elements in the array arr. |

**Pandas**

|  |  |  |
| --- | --- | --- |
| **Method** | **Description** | **Example** |
| head() | Description: Returns the first n rows of a DataFrame. | Example: df.head(5) - Returns the first 5 rows of DataFrame df. |
| tail() | Returns the last n rows of a DataFrame. | Example: df.tail(3) - Returns the last 3 rows of DataFrame df. |
| info() | Provides a summary of a DataFrame's structure. | df.info() - Displays the column names, data types, and memory usage of DataFrame df. |
| describe() | Generates descriptive statistics of a DataFrame. | df.describe() - Computes count, mean, std, min, max, and quartiles of DataFrame df. |
| shape | Returns the dimensions (rows, columns) of a DataFrame. | df.shape - Returns the number of rows and columns in DataFrame df. |
| columns | Returns the column labels of a DataFrame. | df.columns - Returns the column names of DataFrame df. |
| index | Returns the row labels of a DataFrame. | df.index - Returns the row indices of DataFrame df. |
| loc[] | Accesses a group of rows and columns by label(s). | df.loc[3:5, 'column\_name'] - Retrieves rows 3 to 5 and the specified column from DataFrame df. |
| iloc[] | Accesses a group of rows and columns by integer position(s). | df.iloc[2:4, 0:3] - Retrieves rows 2 to 3 and columns 0 to 2 from DataFrame df. |
| dropna() | Drops rows with missing values from a DataFrame. | df.dropna() - Removes rows with any NaN values from DataFrame df. |
| fillna() | Fills missing values in a DataFrame with specified values. | df.fillna(0) - Replaces NaN values in DataFrame df with 0. |
| groupby() | Groups data based on one or more columns. | df.groupby('column\_name') - Groups data in DataFrame df based on the specified column. |
| sort\_values() | Sorts a DataFrame by specified columns. | df.sort\_values('column\_name') - Sorts DataFrame df based on the specified column. |
| merge() | Merges two DataFrames based on a common column. | merged\_df = pd.merge(df1, df2, on='column\_name') - Merges df1 and df2 based on the specified column. |
| pivot\_table() | Creates a spreadsheet-style pivot table. | df.pivot\_table(index='column\_1', columns='column\_2', values='column\_3', aggfunc='mean') - Creates a pivot table based on specified columns and aggregating function. |
| apply() | Applies a function to each element or column of a DataFrame. | df['column\_name'].apply(lambda x: x \* 2) - Applies the lambda function to each element in the specified column. |
| astype() | Converts the data type of one or more columns. | df['column\_name'].astype('int') - Converts the data type of the specified column to integer. |
| to\_csv() | Writes a DataFrame to a CSV file. | df.to\_csv('file.csv', index=False) - Saves DataFrame df to a CSV file without including the index. |
| read\_csv() | Reads a CSV file and returns a DataFrame. | df = pd.read\_csv('file.csv') - Reads the CSV file and assigns its contents to DataFrame df. |

**Matplotlib**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Method** | | **Description** | | **Example** |
| plt.plot() | Plots a line or marker-based graph. | | plt.plot(x, y, 'r-', label='line') | |
| plt.scatter() | Plots a scatter plot. | | plt.scatter(x, y, c='r', label='points') | |
| plt.bar() | Plots a bar graph. | | plt.bar(x, height, width=0.8, color='b', label='bars') | |
| plt.hist() | Plots a histogram. | | plt.hist(data, bins=10, color='g', label='histogram') | |
| plt.pie() | Plots a pie chart. | | plt.pie(values, labels=labels, colors=colors) | |
| plt.xlabel() | Sets the label for the x-axis. | | plt.xlabel('x-axis label') | |
| plt.ylabel() | Sets the label for the y-axis. | | plt.ylabel('y-axis label') | |
| plt.title() | Sets the title of the plot. | | plt.title('Plot Title') | |
| plt.legend() | Adds a legend to the plot. | | plt.legend() | |
| plt.grid() | Adds grid lines to the plot. | | plt.grid(True) | |
| plt.xlim() | Sets the x-axis limits. | | plt.xlim(0, 10) | |
| plt.ylim() | Sets the y-axis limits. | | plt.ylim(0, 100) | |
| plt.xticks() | Sets the ticks on the x-axis. | | plt.xticks([1, 2, 3, 4, 5], ['A', 'B', 'C', 'D', 'E']) | |
| plt.yticks() | Sets the ticks on the y-axis. | | plt.yticks([0, 20, 40, 60, 80, 100], ['0%', '20%', '40%', '60%', '80%', '100%']) | |
| plt.savefig() | Saves the plot as an image file. | | plt.savefig('plot.png') | |
| plt.show() | Displays the plot. | | plt.show() | |
| plt.subplots() | Creates a grid of subplots. | | fig, axes = plt.subplots(nrows, ncols) | |
| plt.imshow() | Displays an image. | | plt.imshow(image) | |
| plt.colorbar() | Adds a colorbar to the plot. | | plt.colorbar() | |
| plt.annotate() | Adds an annotation to a plot. | | plt.annotate('Text', xy=(x, y), xytext=(x\_text, y\_text), arrowprops=dict(facecolor='black', arrowstyle='->')) | |
| plt.fill\_between() | Fills the area between two curves. | | plt.fill\_between(x, y1, y2, color='gray', alpha=0.5) | |
| plt.plot\_date() | Plots data points with dates on the x-axis. | | plt.plot\_date(dates, values, linestyle='-', marker='o') | |
| plt.subplot() | Creates a single subplot within a grid. | | plt.subplot(rows, cols, index) | |
| plt.tight\_layout() | Adjusts the spacing between subplots. | | plt.tight\_layout() | |
| plt.loglog() | Plots data on a logarithmic scale. | | plt.loglog(x, y) | |
| plt.semilogx() | Plots data on a logarithmic scale for the x-axis. | | plt.semilogx(x, y) | |
| plt.semilogy() | Plots data on a logarithmic scale for the y-axis. | | plt.semilogy(x, y) | |

**Scikit Learn**

|  |  |
| --- | --- |
| **Method** | **Description** |
| sklearn.model\_selection | Provides tools for model selection and evaluation, such as train-test split, cross-validation, and hyperparameter tuning. |
| sklearn.preprocessing | Contains utilities for data preprocessing, such as scaling, encoding categorical variables, and imputing missing values. |
| sklearn.feature\_extraction | Offers methods to extract features from raw data, including text and image data. |
| sklearn.linear\_model | Contains various linear models for regression and classification tasks. |
| sklearn.tree | Provides decision tree-based algorithms for regression and classification tasks. |
| sklearn.ensemble | Includes ensemble methods like Random Forest and Gradient Boosting for regression and classification. |
| sklearn.svm | Contains Support Vector Machine (SVM) algorithms for classification and regression. |
| sklearn.cluster | Offers clustering algorithms like K-Means and hierarchical clustering. |
| sklearn.metrics | Provides various evaluation metrics for machine learning models, such as accuracy, precision, recall, and F1-score. |
| sklearn.datasets | Contains several sample datasets for practice and experimentation. |
| sklearn.pipeline | Provides a way to chain multiple machine learning steps together in a single pipeline. |
| sklearn.naive\_bayes | Includes Naive Bayes algorithms for classification tasks. |
| sklearn.neighbors | Contains k-Nearest Neighbors algorithms for classification and regression. |
| sklearn.decomposition | Provides methods for dimensionality reduction, such as Principal Component Analysis (PCA). |
| sklearn.neural\_network | Includes neural network-based models, such as Multi-Layer Perceptron (MLP) for classification and regression. |

**Requests**

|  |  |  |
| --- | --- | --- |
| **Method** | **Description** | **Example** |
| get() | Sends a GET request to the specified URL. | response = requests.get('https://api.example.com') |
| post() | Sends a POST request to the specified URL. | data = {'key': 'value'}\nresponse = requests.post('https://api.example.com', data=data) |
| put() | Sends a PUT request to the specified URL. | data = {'key': 'updated\_value'}\nresponse = requests.put('https://api.example.com', data=data) |
| delete() | Sends a DELETE request to the specified URL. | response = requests.delete('https://api.example.com') |
| head() | Sends a HEAD request to the specified URL. | response = requests.head('https://api.example.com') |
| options() | Sends an OPTIONS request to the specified URL. | response = requests.options('https://api.example.com') |
| patch() | Sends a PATCH request to the specified URL. | data = {'key': 'updated\_value'}\nresponse = requests.patch('https://api.example.com', data=data) |
| request() | Constructs and sends any HTTP request. | response = requests.request('POST', 'https://api.example.com') |
| session() | Creates a persistent session for HTTP requests. | session = requests.session() |
| get\_json() | Sends a GET request and parses the response as JSON. | data = requests.get\_json('https://api.example.com/json\_data') |
| status\_code() | Returns the status code of the response. | code = response.status\_code |
| headers() | Returns the headers of the response. | headers = response.headers |
| cookies() | Returns the cookies of the response. | cookies = response.cookies |
| text() | Returns the response content as a string. | content = response.text |
| json() | Returns the response content as JSON. | json\_data = response.json() |
| content() | Returns the response content as bytes. | content = response.content |
| raise\_for\_status() | Raises an exception if the request is not successful. | response.raise\_for\_status() |
| timeout() | Specifies the maximum time to wait for the response. | response = requests.get('https://api.example.com', timeout=5) |
| headers() | Sets the headers of the request. | headers = {'User-Agent': 'Mozilla/5.0'}\nresponse = requests.get('https://api.example.com', headers=headers) |
| params() | Sets the parameters of the request. | params = {'key': 'value'}\nresponse = requests.get('https://api.example.com', params=params) |
| auth() | Sets the authentication credentials for the request. | response = requests.get('https://api.example.com', auth=('username', 'password')) |

**Beautiful Soup**

|  |  |  |
| --- | --- | --- |
| **Method** | **Description** | **Example** |
| find() | Finds the first matching tag in the parsed HTML. | soup.find('tag\_name') |
| find\_all() | Finds all matching tags in the parsed HTML. | soup.find\_all('tag\_name') |
| select() | Finds tags using CSS selectors. | soup.select('selector') |
| get\_text() | Extracts the text from a tag. | tag.get\_text() |
| get() | Retrieves the value of an attribute from a tag. | tag.get('attribute') |
| parent() | Accesses the parent tag of a tag. | tag.parent |
| children() | Iterates over the direct children of a tag. | tag.children |
| contents() | Returns the contents of a tag as a list. | tag.contents |
| next\_sibling() | Accesses the next sibling tag of a tag. | tag.next\_sibling |
| previous\_sibling() | Accesses the previous sibling tag of a tag. | tag.previous\_sibling |
| find\_parent() | Finds the closest parent tag that matches a criteria. | tag.find\_parent('tag\_name') |
| find\_next\_sibling() | Finds the next sibling tag that matches a criteria. | tag.find\_next\_sibling('tag\_name') |
| find\_previous\_sibling() | Finds the previous sibling tag that matches a criteria. | tag.find\_previous\_sibling('tag\_name') |
| has\_attr() | Checks if a tag has a specific attribute. | tag.has\_attr('attribute') |
| string | Extracts the string within a tag. | tag.string |
| get\_attribute\_list() | Returns a list of attribute names of a tag. | tag.get\_attribute\_list() |
| get\_attribute\_dict() | Returns a dictionary of attribute names and values. | tag.get\_attribute\_dict() |
| decompose() | Removes a tag from the parse tree. | tag.decompose() |

**Opencv**

|  |  |  |
| --- | --- | --- |
| **Method** | **Description** | **Example** |
| cv2.imread() | Reads an image from a file. | img = cv2.imread('image.jpg') |
| cv2.imshow() | Displays an image in a window. | cv2.imshow('Window', img) |
| cv2.cvtColor() | Converts an image from one color space to another. | img\_gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY) |
| cv2.imwrite() | Saves an image to a file. | cv2.imwrite('output.jpg', img) |
| cv2.resize() | Resizes an image to a specified size. | img\_resized = cv2.resize(img, (width, height)) |
| cv2.rectangle() | Draws a rectangle on an image. | cv2.rectangle(img, (x1, y1), (x2, y2), (255, 0, 0), 2) |
| cv2.circle() | Draws a circle on an image. | cv2.circle(img, (x, y), radius, (0, 255, 0), -1) |
| cv2.line() | Draws a line on an image. | cv2.line(img, (x1, y1), (x2, y2), (0, 0, 255), 2) |
| cv2.putText() | Writes text on an image. | cv2.putText(img, 'Text', (x, y), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 0, 255), 2) |
| cv2.bitwise\_and() | Performs bitwise AND operation on images. | img\_result = cv2.bitwise\_and(img1, img2) |
| cv2.bitwise\_or() | Performs bitwise OR operation on images. | img\_result = cv2.bitwise\_or(img1, img2) |
| cv2.bitwise\_not() | Performs bitwise NOT operation on an image. | img\_result = cv2.bitwise\_not(img) |
| cv2.threshold() | Applies a thresholding operation to an image. | ret, img\_thresh = cv2.threshold(img\_gray, 127, 255, cv2.THRESH\_BINARY) |
| cv2.findContours() | Finds contours in a binary image. | contours, hierarchy = cv2.findContours(img\_thresh, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE) |
| cv2.drawContours() | Draws contours on an image. | cv2.drawContours(img, contours, -1, (0, 0, 255), 2) |
| cv2.cvtColor() | Converts an image from one color space to another. | img\_rgb = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB) |
| cv2.split() | Splits an image into its color channels. | b, g, r = cv2.split(img) |
| cv2.merge() | Merges individual color channels into an image. | img\_merged = cv2.merge([b, g, r]) |
| cv2.equalizeHist() | Enhances the contrast of a grayscale image. | img\_eq = cv2.equalizeHist(img\_gray) |
| cv2.GaussianBlur() | Applies Gaussian smoothing to an image. | img\_blur = cv2.GaussianBlur(img, (5, 5), 0) |
| cv2.Canny() | Detects edges in an image using the Canny algorithm. | edges = cv2.Canny(img\_gray, threshold1, threshold2) |