Machine learning can be categorized into different types based on the learning approach, task, and algorithm used. Here are some commonly recognized types of machine learning:

1. Supervised Learning: In supervised learning, the model is trained on labeled data, where the input data is paired with corresponding target labels or outcomes. The goal is to learn a mapping function that can predict labels for new, unseen data. Examples include classification and regression tasks.

2. Unsupervised Learning: Unsupervised learning involves training the model on unlabeled data, where the goal is to discover patterns, structures, or relationships in the data without explicit target labels. It is often used for clustering, dimensionality reduction, and anomaly detection tasks.

3. Semi-Supervised Learning: Semi-supervised learning is a combination of supervised and unsupervised learning. It uses a small amount of labeled data along with a large amount of unlabeled data for training. The model leverages the unlabeled data to improve its performance on the labeled data.

4. Reinforcement Learning: Reinforcement learning focuses on training agents to take actions in an environment to maximize cumulative rewards. The agent learns through trial and error, receiving feedback in the form of rewards or penalties based on its actions. The goal is to learn an optimal policy or strategy.

5. Deep Learning: Deep learning is a subfield of machine learning that utilizes artificial neural networks with multiple layers to learn hierarchical representations of data. Deep learning algorithms, such as deep neural networks, convolutional neural networks (CNNs), and recurrent neural networks (RNNs), are particularly effective in tasks involving large amounts of data, such as image and speech recognition.

6. Transfer Learning: Transfer learning involves leveraging knowledge or models learned from one task or domain to improve performance on another related task or domain. The pre-trained models capture general features and can be fine-tuned or adapted for specific tasks, saving time and resources.

7. Online Learning: Online learning, also known as incremental or streaming learning, involves continuously updating the model as new data becomes available in a sequential manner. The model learns in real-time, adapting to changing patterns and dynamics without requiring access to past data.

8. Ensemble Learning: Ensemble learning combines multiple individual models to make predictions or decisions. By aggregating the predictions of different models, ensemble methods aim to improve overall performance and generalization. Popular ensemble techniques include bagging, boosting, and random forests.

These are some of the types of machine learning commonly used in practice. Machine learning techniques continue to evolve, and researchers are exploring new methods to tackle complex problems and improve the capabilities of intelligent systems.

**Supervised Learning**

1. Classification:

- Logistic Regression: It models the relationship between input features and the probability of belonging to different classes.

- Decision Trees: They create a tree-like model to make decisions based on feature values.

- Random Forest: It is an ensemble method that combines multiple decision trees to improve prediction accuracy.

- Support Vector Machines (SVM): It separates data points into different classes using hyperplanes in a high-dimensional space.

- Naive Bayes: It applies Bayes' theorem with the assumption of independence between features.

- k-Nearest Neighbors (k-NN): It classifies data points based on the majority vote of their nearest neighbors.

2. Regression:

- Linear Regression: It fits a linear relationship between input features and the target variable.

- Polynomial Regression: It extends linear regression by introducing polynomial terms to capture nonlinear relationships.

- Support Vector Regression (SVR): It applies SVM principles to regression tasks.

- Random Forest Regression: It utilizes a random forest ensemble for regression problems.

- Gradient Boosting Regression: It combines weak prediction models in a boosting framework to create a strong regression model.

- Neural Networks: They can be used for regression tasks by training deep learning models with appropriate architectures.

3. Ordinal Regression:

- Ordinal Logistic Regression: It extends logistic regression to handle ordinal target variables.

- Support Vector Ordinal Regression: It applies SVM principles to ordinal regression tasks.

- Cumulative Link Models: They model the cumulative probabilities of different ordinal classes.

4. Multi-label Classification:

- Binary Relevance: It trains separate binary classifiers for each label and combines their predictions.

- Label Powerset: It treats each combination of labels as a distinct class and trains a classifier on the transformed data.

- Classifier Chains: It creates a chain of binary classifiers where each classifier considers the predictions of previous classifiers.

5. Time Series Forecasting:

- Autoregressive Integrated Moving Average (ARIMA): It models time series data by incorporating autoregression, differencing, and moving average components.

- Exponential Smoothing Methods: These methods provide different smoothing techniques to forecast time series data, such as Simple Exponential Smoothing, Holt's Linear Exponential Smoothing, and Holt-Winters' Seasonal Exponential Smoothing.

- Recurrent Neural Networks (RNN): They are effective for sequential data analysis, allowing the modeling of temporal dependencies in time series data.

- Long Short-Term Memory (LSTM) Networks: A type of RNN that can capture long-term dependencies and has shown success in time series forecasting.

6. Anomaly Detection:

- Density-Based Anomaly Detection: It identifies anomalies based on deviations from the expected data density.

- Support Vector Data Description (SVDD): It uses SVM principles to create a model that captures the normal behavior of data, allowing the detection of deviations.

- Isolation Forest: It isolates anomalies by constructing binary trees and identifying instances with shorter average path lengths.

- One-Class SVM: It separates normal instances from outliers using SVM principles with a focus on capturing normal behavior.

These algorithms provide a starting point for solving different types of supervised learning problems. Depending on the specific problem and dataset characteristics, other algorithms and variations may also be suitable.

**Examples**

1. Classification:

- Email Spam Detection: Classify incoming emails as either spam or legitimate based on their content and characteristics.

- Image Classification: Identify objects or scenes in images, such as recognizing different animal species or categorizing handwritten digits.

- Sentiment Analysis: Determine the sentiment (positive, negative, or neutral) expressed in text, such as analyzing customer reviews or social media posts.

- Disease Diagnosis: Classify medical images or patient data to assist in diagnosing diseases or medical conditions.

2. Regression:

- Housing Price Prediction: Predict the selling prices of houses based on features like area, number of bedrooms, location, etc.

- Stock Market Forecasting: Predict future stock prices or market trends based on historical price and trading data.

- Demand Prediction: Estimate the demand for a product or service based on factors like price, time, promotions, and historical sales data.

- Energy Consumption Forecasting: Predict future energy consumption levels for efficient energy management and resource planning.

3. Ordinal Regression:

- Rating Prediction: Predict user ratings on a scale (e.g., 1 to 5 stars) for products, movies, or services based on user preferences and characteristics.

- Survey Response Analysis: Analyze survey responses on an ordered scale to understand customer satisfaction, product preferences, or user feedback.

- Academic Performance Ranking: Rank students based on their academic performance using relevant features such as exam scores, attendance, and participation.

4. Multi-label Classification:

- Document Categorization: Categorize documents into multiple topics or themes, such as classifying news articles into various categories.

- Image Tagging: Assign multiple tags or labels to images based on the objects, scenes, or concepts depicted in the images.

- Multi-label Sentiment Analysis: Analyze text to identify and classify multiple sentiments expressed simultaneously, such as detecting emotions in social media posts.

5. Time Series Forecasting:

- Stock Market Prediction: Forecast future stock prices or market trends based on historical price and trading data.

- Weather Forecasting: Predict weather conditions, temperature, precipitation, and other meteorological factors for specific locations and time periods.

- Sales Demand Forecasting: Forecast future sales or demand patterns for products or services to optimize inventory management and supply chain operations.

6. Anomaly Detection:

- Fraud Detection: Identify suspicious or fraudulent activities in financial transactions, credit card usage, or online transactions.

- Network Intrusion Detection: Detect anomalous behavior or intrusions in computer networks to enhance cybersecurity.

- Outlier Detection: Identify rare events or outliers in datasets, such as detecting manufacturing defects or unusual patterns in sensor data.

These examples demonstrate the wide range of applications where supervised learning techniques are employed to solve various real-world problems effectively.

**Unsupervised Learning**

1. Clustering:

- K-means: Partition data into k clusters based on similarity, where k is a predetermined number. Applications include customer segmentation and image compression.

- Hierarchical Clustering: Build a hierarchy of clusters by recursively merging or splitting them based on similarity. It can be used for taxonomy construction or gene expression analysis.

- DBSCAN (Density-Based Spatial Clustering of Applications with Noise): Discover clusters of varying shapes and densities in data, suitable for spatial data clustering and anomaly detection.

2. Dimensionality Reduction:

- Principal Component Analysis (PCA): Identify the most important components in data by projecting it onto a lower-dimensional space. Applications include data visualization and feature extraction.

- t-SNE (t-Distributed Stochastic Neighbor Embedding): Visualize high-dimensional data in a lower-dimensional space, often used for visualizing clusters or data similarity.

- Autoencoders: Train neural networks to learn efficient representations of input data by compressing and decompressing it. Applications include feature learning and denoising.

3. Association Rule Mining:

- Apriori Algorithm: Discover frequent itemsets and association rules from transactional data. It is used in market basket analysis to find relationships between items that are frequently purchased together.

- FP-Growth (Frequent Pattern-Growth): Efficiently mine frequent itemsets and association rules by building a compact data structure called an FP-tree.

4. Anomaly Detection:

- Isolation Forest: Identify anomalies by isolating them in a binary tree structure based on feature values.

- Local Outlier Factor (LOF): Measure the local density deviation of data points to detect outliers.

- One-Class SVM: Construct a boundary around normal instances and identify outliers based on their distance from the boundary.

5. Generative Models:

- Gaussian Mixture Models (GMM): Model data as a combination of Gaussian distributions to learn underlying patterns and generate new data samples.

- Variational Autoencoders (VAE): Generate new data samples by learning the underlying latent space representation of the data.

- Generative Adversarial Networks (GANs): Generate new data samples by training a generator network to produce data that is indistinguishable from real data.

6. Frequent Pattern Mining:

- Eclat: Find frequent itemsets without generating candidate itemsets, suitable for large-scale transactional data analysis.

- PrefixSpan: Discover sequential patterns in sequential data, often used in market basket analysis and web clickstream analysis.

These examples illustrate the diverse range of applications that unsupervised learning can address, such as clustering similar data points, reducing high-dimensional data, discovering associations, detecting anomalies, and generating new data samples.

**Examples**

1. Clustering:

- Customer Segmentation: Divide customers into groups based on their purchasing behaviors or preferences.

- Image Compression: Reduce the file size of images while preserving important visual information.

- Document Clustering: Group similar documents together based on their content or topics.

2. Dimensionality Reduction:

- Data Visualization: Reduce high-dimensional data to a lower dimension for visualization purposes while preserving important patterns and structures.

- Feature Extraction: Identify a smaller set of meaningful features that capture the most relevant information in the data.

- Noise Reduction: Remove noise or irrelevant features from the data to improve model performance.

3. Association Rule Mining:

- Market Basket Analysis: Discover associations between items frequently purchased together in a transactional dataset.

- Recommender Systems: Generate personalized recommendations by finding common item associations among users.

4. Anomaly Detection:

- Fraud Detection: Identify fraudulent or anomalous activities in financial transactions or online behaviors.

- Network Intrusion Detection: Detect unusual patterns or attacks in network traffic to enhance cybersecurity.

- Equipment Failure Prediction: Identify anomalies in sensor data to predict potential failures or malfunctions.

5. Generative Models:

- Data Generation: Generate synthetic data samples that resemble the characteristics of the original dataset, useful for data augmentation or simulation purposes.

- Image Synthesis: Generate new images that share similar characteristics with a given set of training images.

- Text Generation: Create coherent and contextually relevant sentences or paragraphs of text.

6. Frequent Pattern Mining:

- Market Basket Analysis: Discover frequently co-occurring items in transactional data to uncover purchasing patterns and make business decisions.

- Web Clickstream Analysis: Identify frequently occurring navigation patterns in web logs to optimize website design and user experience.

- DNA Sequence Analysis: Find frequent subsequences or patterns in DNA sequences to understand genetic relationships and mutations.

Unsupervised learning techniques play a crucial role in extracting meaningful information, finding hidden structures, and discovering patterns in data where the target labels or outcomes are unknown or unavailable.

**Deep Learning**

1. Convolutional Neural Networks (CNNs):

- Algorithm: CNNs are composed of convolutional layers that extract spatial hierarchies of features and pooling layers that reduce the spatial dimensions. They are often followed by fully connected layers for classification.

- Examples and Problems Solved:

- Image Classification: Classify images into different categories, such as recognizing objects in photographs or distinguishing between handwritten digits.

- Object Detection: Detect and localize objects within images or video frames.

- Image Segmentation: Segment images into different regions or objects based on their visual features.

2. Recurrent Neural Networks (RNNs):

- Algorithm: RNNs have recurrent connections that enable them to process sequential data by capturing temporal dependencies.

- Examples and Problems Solved:

- Natural Language Processing (NLP): Tasks such as language modeling, machine translation, sentiment analysis, and named entity recognition.

- Speech Recognition: Convert spoken language into written text.

- Time Series Analysis: Predict future values or patterns in time-dependent data, such as stock market forecasting or weather prediction.

3. Generative Adversarial Networks (GANs):

- Algorithm: GANs consist of two components: a generator network that produces synthetic samples, and a discriminator network that distinguishes between real and synthetic samples. They compete and improve together through an adversarial training process.

- Examples and Problems Solved:

- Image Generation: Generate realistic images that resemble real-world data, such as creating new images of human faces or generating artistic images.

- Data Augmentation: Generate additional training data to improve the performance and generalization of other models.

- Text-to-Image Synthesis: Generate images based on textual descriptions.

4. Transformers:

- Algorithm: Transformers employ self-attention mechanisms to capture global dependencies and relationships between different elements in the input sequence.

- Examples and Problems Solved:

- Machine Translation: Translate text or speech from one language to another.

- Question-Answering Systems: Answer questions based on given context or documents.

- Text Summarization: Generate concise summaries of long documents or articles.

5. Autoencoders:

- Algorithm: Autoencoders consist of an encoder network that maps input data to a lower-dimensional representation, and a decoder network that reconstructs the input from the lower-dimensional representation.

- Examples and Problems Solved:

- Dimensionality Reduction: Extract meaningful and compact representations of input data for visualization or downstream tasks.

- Anomaly Detection: Detect outliers or anomalies by comparing the reconstructed input with the original input.

- Image Denoising: Remove noise or artifacts from images to improve their quality.

These are just a few examples of subtypes of deep learning and the problems they solve. Deep learning techniques have shown remarkable success in various domains, including computer vision, natural language processing, speech recognition, and sequential data analysis.

**Reinforcement Learning**

1. Value-Based Methods:

- Q-Learning: It learns an action-value function that estimates the expected cumulative rewards for each state-action pair. It uses an exploration-exploitation strategy to find an optimal policy.

- Deep Q-Networks (DQN): It combines Q-Learning with deep neural networks to handle high-dimensional state spaces. It utilizes experience replay and target networks for more stable training.

- Examples and Problems Solved:

- Game Playing: Achieving superhuman performance in games like chess, Go, or Atari games by learning optimal strategies.

- Robotics Control: Training robotic agents to perform complex tasks and manipulate objects in various environments.

2. Policy-Based Methods:

- REINFORCE: It directly learns a parameterized policy by estimating the expected cumulative reward and updates the policy using gradient ascent.

- Proximal Policy Optimization (PPO): It aims to improve stability and sample efficiency compared to REINFORCE by optimizing the policy in a conservative way.

- Examples and Problems Solved:

- Autonomous Driving: Training self-driving cars to navigate roads and make safe driving decisions based on policy learning.

- Robotic Manipulation: Teaching robots to perform precise movements and manipulation tasks using learned policies.

3. Actor-Critic Methods:

- Advantage Actor-Critic (A2C): It combines the advantages of policy-based and value-based methods by using a value function as a critic to estimate advantages and update the policy accordingly.

- Deep Deterministic Policy Gradient (DDPG): It extends actor-critic methods to continuous action spaces using deterministic policies and Q-value functions.

- Examples and Problems Solved:

- Continuous Control: Controlling robotic arms, drones, or other systems with continuous action spaces to achieve precise movements and control.

4. Model-Based Methods:

- Monte Carlo Tree Search (MCTS): It builds a search tree by simulating and expanding possible future states to find an optimal action sequence.

- Model Predictive Control (MPC): It utilizes a predictive model of the environment to plan and optimize actions over a finite time horizon.

- Examples and Problems Solved:

- Game Playing: Planning and decision-making in games like chess, Go, or real-time strategy games.

- Resource Management: Optimizing resource allocation, scheduling, and control in dynamic environments.

5. Multi-Agent Reinforcement Learning:

- Independent Q-Learning: Each agent learns independently and aims to maximize its own rewards without considering the actions or policies of other agents.

- Deep Multi-Agent Reinforcement Learning (MARL): Extends reinforcement learning techniques to scenarios where multiple agents interact and learn simultaneously.

- Examples and Problems Solved:

- Multi-Agent Coordination: Cooperation, competition, or negotiation tasks involving multiple agents, such as multi-robot systems or multi-player games.

These are just a few examples of subtypes of reinforcement learning and the problems they solve. Reinforcement learning techniques are applicable in various domains where agents learn to interact with environments, make decisions, and optimize long-term rewards.

**Natural Language Processing NLP**

Performing NLP (Natural Language Processing) involves a series of steps to process and analyze text or speech data. Here's a general outline of how to perform NLP:

1. Text Preprocessing:

- Tokenization: Splitting text into individual words or tokens.

- Cleaning: Removing unnecessary characters, punctuation, or special symbols.

- Lowercasing: Converting text to lowercase to ensure consistency.

- Stopword Removal: Removing common words (e.g., "the," "is," "and") that do not carry significant meaning.

- Stemming/Lemmatization: Reducing words to their base or root form (e.g., "running" to "run").

2. Feature Extraction:

- Bag-of-Words: Representing text as a collection of word frequencies or presence/absence indicators.

- TF-IDF (Term Frequency-Inverse Document Frequency): Assigning weights to words based on their frequency in a document and rarity across the corpus.

- Word Embeddings: Representing words as dense vector representations capturing semantic relationships (e.g., Word2Vec, GloVe, or fastText).

3. NLP Tasks:

- Sentiment Analysis: Determining the sentiment (positive, negative, or neutral) expressed in text.

- Named Entity Recognition (NER): Identifying and classifying named entities such as names, locations, organizations, etc.

- Part-of-Speech (POS) Tagging: Assigning grammatical tags to words (e.g., noun, verb, adjective).

- Text Classification: Assigning predefined categories or labels to text (e.g., spam detection, topic classification).

- Machine Translation: Converting text from one language to another.

- Question Answering: Providing answers to questions based on a given context or document.

- Topic Modeling: Discovering latent topics or themes in a collection of documents.

4. Machine Learning or Deep Learning:

- Selecting a suitable algorithm or model based on the specific NLP task.

- Training the model using annotated or labeled data for supervised learning tasks.

- Fine-tuning pre-trained models or using transfer learning for improved performance.

- Evaluating the model's performance using appropriate metrics.

5. Post-processing and Evaluation:

- Post-process the model's outputs based on task-specific requirements.

- Evaluate the performance of the model using metrics such as accuracy, precision, recall, F1-score, etc.

- Iterate and refine the process by adjusting preprocessing steps, feature representations, or models based on the evaluation results.

It's important to note that the specific steps and techniques involved in NLP may vary depending on the task, data, and available resources. Additionally, there are several NLP libraries and frameworks (e.g., NLTK, spaCy, scikit-learn, TensorFlow, PyTorch) that provide pre-built tools and models to simplify the NLP process.

Sentiment analysis aims to determine the sentiment expressed in a given piece of text (positive, negative, or neutral). Here's how you can perform sentiment analysis:

1. Text Preprocessing:

- Input Text: "I absolutely loved the movie! The acting was fantastic, and the plot kept me engaged throughout."

- Tokenization: Split the text into individual words: ["I", "absolutely", "loved", "the", "movie", "!", "The", "acting", "was", "fantastic", ",", "and", "the", "plot", "kept", "me", "engaged", "throughout", "."]

- Cleaning: Remove special characters and punctuation: ["I", "absolutely", "loved", "the", "movie", "The", "acting", "was", "fantastic", "and", "the", "plot", "kept", "me", "engaged", "throughout"]

- Lowercasing: Convert all words to lowercase: ["i", "absolutely", "loved", "the", "movie", "the", "acting", "was", "fantastic", "and", "the", "plot", "kept", "me", "engaged", "throughout"]

- Stopword Removal: Remove common words like "the", "was", "and": ["i", "absolutely", "loved", "movie", "acting", "fantastic", "plot", "kept", "engaged", "throughout"]

- Stemming/Lemmatization: Reduce words to their base or root form: ["i", "absolutely", "love", "movi", "act", "fantast", "plot", "kept", "engag", "throughout"]

2. Feature Extraction:

- Bag-of-Words: Create a vector representation of the text based on word frequencies or presence/absence indicators:

{"i": 1, "absolutely": 1, "love": 1, "movi": 1, "act": 1, "fantast": 1, "plot": 1, "kept": 1, "engag": 1, "throughout": 1}

- TF-IDF: Assign weights to words based on their frequency in the document and rarity across the corpus.

3. Sentiment Analysis:

- Machine Learning/Deep Learning Model: Train a classifier (e.g., Support Vector Machines, Naive Bayes, or a deep learning model) using labeled data. The labeled data contains texts with corresponding sentiment labels (positive, negative, or neutral).

- Training: Feed the preprocessed text and corresponding sentiment labels into the model for training.

- Evaluation: Assess the trained model's performance using evaluation metrics such as accuracy, precision, recall, or F1-score.

4. Prediction:

- Input New Text: "The movie was a disappointment. The acting was terrible, and the plot was dull."

- Preprocess the new text using the same steps as mentioned earlier.

- Apply the trained model to predict the sentiment of the new text.

The result of sentiment analysis could be, for example, "Negative" for the input text "The movie was a disappointment. The acting was terrible, and the plot was dull."

This example illustrates the general steps involved in performing sentiment analysis using NLP techniques. Depending on the specific task and available resources, you may need to adjust the steps, explore different preprocessing techniques, or experiment with various machine learning or deep learning algorithms to achieve optimal results.

**Example Program**

To solve the problem and calculate the given metrics from a given text, you can use various NLP libraries and techniques. Here's a step-by-step approach to solving the problem using Python and the NLTK library:

1. Import the required libraries:

import nltk

from textstat import flesch\_reading\_ease, syllable\_count

from nltk.sentiment import SentimentIntensityAnalyzer

from nltk.tokenize import sent\_tokenize, word\_tokenize

2. Define the given text:

text = "Your given text goes here."

3. Perform sentiment analysis and obtain the sentiment scores:

sia = SentimentIntensityAnalyzer()

sentiment\_scores = sia.polarity\_scores(text)

positive\_score = sentiment\_scores['pos']

negative\_score = sentiment\_scores['neg']

polarity\_score = sentiment\_scores['compound']

4. Perform subjectivity analysis:

subjectivity\_score = sentiment\_scores['neu'] + sentiment\_scores['pos']

5. Tokenize the text into sentences and words:

sentences = sent\_tokenize(text)

words = word\_tokenize(text)

6. Calculate the average sentence length:

avg\_sentence\_length = len(words) / len(sentences)

7. Calculate the percentage of complex words:

complex\_word\_count = 0

for word in words:

if syllable\_count(word) > 2:

complex\_word\_count += 1

percentage\_complex\_words = (complex\_word\_count / len(words)) \* 100

8. Calculate the FOG index:

fog\_index = 0.4 \* (avg\_sentence\_length + percentage\_complex\_words)

9. Calculate the average number of words per sentence:

avg\_words\_per\_sentence = len(words) / len(sentences)

10. Calculate the complex word count:

complex\_word\_count = sum(syllable\_count(word) > 2 for word in words)

11. Calculate the word count:

word\_count = len(words)

12. Calculate the average word length:

avg\_word\_length = sum(len(word) for word in words) / word\_count

13. Identify personal pronouns:

pronouns = ['I', 'you', 'he', 'she', 'it', 'we', 'they', 'me', 'you', 'him', 'her', 'us', 'them']

personal\_pronouns = sum(word.lower() in pronouns for word in words)

Now you have the calculated metrics based on the given text. You can print or utilize these values as needed. Remember to have NLTK and the textstat library installed to access the necessary functions.

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) |