ML4Cyber — Lab 4 API Reference

Pandas (`pandas`)

Docs: `https://pandas.pydata.org/docs/`

- Use `pandas.read_csv(...)` (`https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html`) to load a CSV into a `DataFrame`.
- Use `df.copy()` (`https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.copy.html`) to create a (deep) copy of a `DataFrame` before modifying it.
- Use `df.drop(...)` (`https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.drop.html`) to remove columns or rows by label.
- Use `s.astype(...)` (`https://pandas.pydata.org/docs/reference/api/pandas.Series.astype.html`) to cast a `Series` to a new dtype (e.g., bytes → string).
- Use `s.value_counts()` (`https://pandas.pydata.org/docs/reference/api/pandas.Series.value_counts.html`) to count category frequencies.
- Use `df.values` (`https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.values.html`) to get the underlying NumPy array for ML preprocessing.

NumPy (`numpy`)

Docs: `https://numpy.org/doc/stable/`

- Use `numpy.array(...)` (`https://numpy.org/doc/stable/reference/generated/numpy.array.html`) to create arrays (or to enforce an array type).
- Use `numpy.absolute(x)` / `numpy.abs(x)` (`https://numpy.org/doc/stable/reference/generated/numpy.absolute.html`) for elementwise absolute values.
- Use `numpy.nonzero(x)` (`https://numpy.org/devdocs/reference/generated/numpy.nonzero.html`) to get indices where elements are non-zero / `True`.
- Use `numpy.squeeze(x)` (`https://numpy.org/devdocs/reference/generated/numpy.squeeze.html`) to remove singleton dimensions.
- Use `x[:, None]` (a.k.a. `newaxis`) to add a singleton dimension for broadcasting; see NumPy indexing basics (`https://numpy.org/devdocs/user/absolute_beginners.html`).

scikit-learn (`sklearn`)

Docs: `https://scikit-learn.org/stable/`

- Call `sklearn.datasets.fetch_kddcup99(...)` (`https://scikit-learn.org/stable/modules/generated/sklearn.datasets.fetch_kddcup99.html`) to load the KDDCup'99 dataset.
- Call `train_test_split(X, y, ...)` (`https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html`) to create train/test partitions.

- Call `MinMaxScaler().fit_transform(X)` (`https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html`) to scale features into a fixed range (commonly $[0, 1]$).
- Call `StandardScaler().fit_transform(X)` (`https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html`) to standardize features (zero mean, unit variance).
- Call `PCA(n_components=...).fit_transform(X)` (`https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html`) for linear dimensionality reduction.
- Call `KernelPCA(n_components=..., kernel=...).fit_transform(X)` (`https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.KernelPCA.html`) for nonlinear PCA via kernels.
- Call `IncrementalPCA(...).fit_transform(X)` (`https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.IncrementalPCA.html`) for batch-wise PCA on larger datasets.
- Call `TSNE(n_components=2, ...).fit_transform(X)` (`https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html`) to produce a 2D embedding for visualization.

Matplotlib (`matplotlib`)

Docs: `https://matplotlib.org/stable/`

- Call `plt.figure(...)` (Pyplot tutorial: `https://matplotlib.org/stable/tutorials/pyplot.html`) to create a figure.
- Call `fig.add_subplot(...)` (`https://matplotlib.org/stable/api/_as_gen/matplotlib.figure.Figure.add_subplot.html`) to add axes (including 3D axes with `projection='3d'`).
- Call `ax.scatter(x, y, ...)` (`https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.scatter.html`) for scatter plots.
- Call `plt.show()` (`https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.show.html`) to display the figure.
- For 3D plotting, see `mplot3d` toolkit docs (`https://matplotlib.org/stable/api/toolkits/mplot3d.html`).

Python Standard Library

- Use `time.time()` and `time.sleep(...)` (time module docs: `https://docs.python.org/uk/3.8/library/time.html`) for simple timing and pacing.
- Use `@contextlib.contextmanager` (contextlib docs: `https://docs.python.org/3/library/contextlib.html`) to create lightweight context managers (e.g., for timing blocks).