Data Science Libraries

Data Science Libraries

Hier eine Übersicht über gängige Data Science Pakete, wie sie installiert und verwendet werden.

Library Name	Paket Name (pip install <name>)	Code-Beispiel	Beschreibung
Jupyter	jupyter, jupyterlab	`$ jupyter notebook, $ jupyter lab  # startet Notebook oder Lab`	Führt eine interaktive Entwicklungsumgebung ein, die im Browser läuft. Erlaubt Ausführen von Code sowie Eingabe von formatiertem Text (Markdown). Notebooks haben eine eigene Endung `.ipynb`. Zellen können einzeln und wiederholt ausgeführt werden und die Ausgabe bleibt gespeichert.

Numpy	numpy	`import numpy as np`
		`x = np.array([1,2,3,4]  # neues array anlegen`
		`y = -x  # neues array aus bestehendem`
		`x + y  # elementweise Operation`
		`x.mean()  # Aggregationsfunktionen (sum,min,max,…)`
		`x.dtype.  # Datentyp Attribute`
		`z = np.random.randn(3,3) # Zufallszahlen generieren (3x3 Matrix)`
		`z[1:3,1:3]  # Zugriff via Slice (2x2 Sub-Matrix)`
		`np.log(x)  # elementweise Funktionen`
			Führt den Datentyp `ndarray` ein und erlaubt Arbeiten mit multidimensionalen arrays. Arrays sind typisiert und besitzen immer einen Datentyp. Erlaubt einfachen Zugriff auf mehrere Dimensionen über Slices. Führt elementweise Operationen als Standardverhalten ein. Numpy-Operationen sind um mehrere Größenordnungen schneller als Python.

Pandas	pandas	`import pandas as pd`
		`df = pd.read_csv('data.csv', sep="\t")  # Einlesen von CSV/TSV`
		`df.head(10)  # Ersten n Elemente`
		`df.info()  # Metainfo Spalten(typen)`
		`df.describe()  # Statistiken pro Spalte`
		`df["A"], df.loc[:, „A"]  # Spaltenzugriff ü. Label`
		`df.loc["label"]  # Zeilenzugriff ü. Label`
		`df["C"] = df["A"] * df["B"]  # Anlegen neuer Spalte`
		`df.groupby(„A").sum()  # Gruppieren+Aggregieren`
		`df.groupby(„A").agg([„sum", „mean"])  # Gruppieren+Aggregieren`
		`df.plot(), df.plot(kind="bar")  # Spalten visualisieren`
			Führt den Datentyp `DataFrame` und `Series` ein und erlaubt Arbeiten auf tabellarischen Daten. Erlaubt zeilen- sowie spaltenbasierten Zugriff, Manipulation der Tabelle und bietet umfangreiche Transformationsmethoden. Baut auf numpy auf und ergänzt `arrays` um Label. Besitzt eigene Plotting-Funktionen.

Library Name	Paket Name (pip install <name>)	Code-Beispiel	Beschreibung
Matplotlib	matplotlib	`import matplotlib.pyplot as plt` `plt.plot(x,y)` # Linienplot von (x,y) `plt.scatter(x,y)` # Scatterplot von (x,y) `plt.hist(x)` # Histogramm von x `plt.figure(figsize=(8,8))` # neue Grafik mit Maßen 8x8 `fig, ax = plt.subplots(2,2)` # neue Grafik m. 2x2 Plots `plt.title(„Titeltext“)` # Titel `plt.xlabel(„…“)` # x-Achsenbeschriftung `plt.ylabel(„…“)` # y-Achsenbeschriftung `fig.savefig(„name.png“)` # Grafik speichern	Führt Visualisierungsmethoden für kategorische und numerische Variablen ein. Erlaubt einfache und komplexe Visualisierungen. Kann über weitere Parameter und Methoden 100% angepasst werden (`linestyle, marker, color`). Fügt Plots so lange der aktuellen Grafik hinzu, bis diese geschlossen wird.
Seaborn	seaborn	`import seaborn as sns` `sns.load_dataset(“iris”)` # Beispieldatensatz `sns.set_style(„white“)` # Ästhetik definieren `sns.relplot(df, kind=“line“)` # Linienplot f. alle Spalten `sns.relplot(data=df, x=“A“, y=“B“)` # Scatterplot für A & B `sns.displot(df)` # Histogramm f. alle Spalten `sns.displot(data=df, x=“A“, kind=“kde“)` # Densityplot für A `sns.kdeplot(data=df, x=“A“, y=“B“)` # 2D-Densityplot für A & B `sns.catplot(data=df, kind=“box“)` # Boxplot für alle Spalten `sns.displot(data=df, x=“A“, col=“B“)` # Histogramm für A pro B	Führt mächtige Visualisierungsmethoden für kategorische und numerische Variablen ein. Arbeitet mit pandas `DataFrames` zusammen und erlaubt Auswahl der gewünschten Spalten über deren Namen. Bietet high-level Figureplots und low-level Axenplots.

scikit-learn	scikit-learn	`from sklearn import linear_model`
		`reg = linear_model.LinearRegression()` # Linear Regression Modell
		`reg.fit(X, y)` # trainiert das Modell
		`from sklearn import tree`
		`clf = tree.DecisionTreeClassifier()` # Entscheidungsbaum Modell
		`clf = clf.fit(X, y)` # trainiert das Modell
		`from sklearn.cluster import KMeans`
		`kmeans = KMeans(n_clusters=2)` # KMeans Modell m. 2 Cluster
		`kmeans.fit(X)` # trainiert das Modell
		`kmeans.predict(X)` # Vorhersage für neue Daten
		`from sklearn import preprocessing`
		`X_train, X_test, y_train, y_test = train_test_split(`
		`  X, y, test_size=0.4)` # Train- und Testset
		`from sklearn.model_selection import cross_val_score`
		`clf = svm.SVC(kernel='linear', C=1, random_state=42)`
		`scores = cross_val_score(clf, X, y, cv=5)` # Cross-validation
			Führt eine sehr verständliche und einfache API für Machine Learning ein. Erlaubt ein einheitliches Trainieren einer Großzahl von Machine Learning Algorithmen sowohl für supervised als auch unsupervised learning. Bietet darüber hinaus wertvolle Methoden für das Preprocessing, die Modellevaluierung sowie für Hyperparameteroptimierung.