Python Libraries Commands for Data Analysis and Machine Learning

This document provides a quick reference guide to essential commands and their usage in Pandas, Numpy, Seaborn, and Scikit-learn libraries.

# Pandas

import pandas as pd # Import pandas library

df = pd.read\_csv('filename.csv') # Load data from a CSV file into a DataFrame

df.head() # Display the first few rows of the DataFrame

df.info() # Get a concise summary of the DataFrame

df.describe() # Generate descriptive statistics of the DataFrame

df['column\_name'] # Access a specific column

df[['col1', 'col2']] # Access multiple columns

df.loc[row\_index] # Access a row by index

df.iloc[row\_index] # Access a row by integer position

df.isnull().sum() # Check for missing values in the DataFrame

df.dropna() # Remove rows with missing values

df.fillna(value) # Fill missing values with a specific value

df['column'] = df['column'].astype('int') # Convert column data type

df.groupby('column').mean() # Group by column and calculate the mean

df.sort\_values('column') # Sort DataFrame by a specific column

df.merge(other\_df, on='key') # Merge DataFrames on a key column

pd.concat([df1, df2]) # Concatenate DataFrames

df['date'] = pd.to\_datetime(df['date']) # Convert column to datetime

df.resample('M').mean() # Resample time-series data to monthly frequency

# Numpy

import numpy as np # Import numpy library

arr = np.array([1, 2, 3]) # Create a numpy array

arr.shape # Get the shape of the array

arr.reshape((rows, cols)) # Reshape the array

arr.mean() # Calculate the mean of the array

arr.std() # Calculate the standard deviation of the array

np.median(arr) # Calculate the median of the array

np.arange(start, stop, step) # Generate an array with a range of values

np.linspace(start, stop, num) # Generate an array with evenly spaced values

np.random.rand(rows, cols) # Generate an array with random values

arr + arr2 # Element-wise addition

arr \* arr2 # Element-wise multiplication

np.dot(arr1, arr2) # Dot product of two arrays

np.linalg.inv(matrix) # Inverse of a matrix

arr[arr > 0] # Boolean indexing to filter array elements

# Seaborn

import seaborn as sns # Import seaborn library

sns.set(style='darkgrid') # Set the style of seaborn plots

sns.histplot(data=df, x='column') # Create a histogram

sns.scatterplot(data=df, x='x\_col', y='y\_col') # Create a scatter plot

sns.lineplot(data=df, x='x\_col', y='y\_col') # Create a line plot

sns.barplot(data=df, x='x\_col', y='y\_col') # Create a bar plot

sns.boxplot(data=df, x='x\_col', y='y\_col') # Create a box plot

sns.violinplot(data=df, x='x\_col', y='y\_col') # Create a violin plot

sns.heatmap(data=df.corr(), annot=True) # Create a heatmap of correlation matrix

sns.pairplot(data=df) # Create a pair plot

sns.jointplot(data=df, x='x\_col', y='y\_col', kind='scatter') # Create a joint plot

# Scikit-learn

from sklearn.model\_selection import train\_test\_split # Import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) # Split data into training and testing sets

from sklearn.preprocessing import StandardScaler # Import StandardScaler

scaler = StandardScaler() # Create a StandardScaler object

X\_train\_scaled = scaler.fit\_transform(X\_train) # Fit and transform the training data

X\_test\_scaled = scaler.transform(X\_test) # Transform the testing data

from sklearn.linear\_model import LinearRegression # Import LinearRegression

model = LinearRegression() # Create a LinearRegression model

model.fit(X\_train, y\_train) # Fit the model to the training data

y\_pred = model.predict(X\_test) # Predict on the test data

from sklearn.metrics import mean\_squared\_error # Import mean\_squared\_error

mse = mean\_squared\_error(y\_test, y\_pred) # Calculate mean squared error

from sklearn.metrics import accuracy\_score # Import accuracy\_score

accuracy = accuracy\_score(y\_test, y\_pred) # Calculate accuracy score

from sklearn.ensemble import RandomForestClassifier # Import RandomForestClassifier

clf = RandomForestClassifier() # Create a RandomForestClassifier model

clf.fit(X\_train, y\_train) # Fit the classifier to the training data

y\_pred\_class = clf.predict(X\_test) # Predict on the test data

from sklearn.decomposition import PCA # Import PCA

pca = PCA(n\_components=2) # Create a PCA object with 2 components

X\_pca = pca.fit\_transform(X) # Fit and transform the data with PCA

from sklearn.cluster import KMeans # Import KMeans

kmeans = KMeans(n\_clusters=3) # Create a KMeans object with 3 clusters

y\_kmeans = kmeans.fit\_predict(X) # Fit and predict clusters