

Key Features of Pandas for Data Manipulation and Analysis

DataFrame Creation:

Create dataframes from dictionaries, lists, CSV files, Excel files, and many other formats.

python

```
import pandas as pd
df = pd.DataFrame({
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25, 30, 35],
    'City': ['NY', 'LA', 'Chicago']
})
```

Accessing and Selecting Data:

Use `.loc[]` (label-based) and `.iloc[]` (integer position based) indexing to select rows and columns.

python

```
df.loc[0, 'Name'] # Access first row, 'Name' column
df.iloc[0:2, 1] # First two rows, second column
```

Adding and Removing Columns:

Add new columns or remove existing ones easily.

python

```
df['Country'] = 'USA'
df.drop('City', axis=1, inplace=True)
```

Filtering and Querying:

Filter rows using conditions:

python

```
df[df['Age'] > 28]
```

Handling Missing Data:

Methods like `.isna()`, `.fillna()`, `.dropna()` allow identifying and managing missing values.

python

```
df.fillna(0, inplace=True)
```

Aggregation and Grouping:

Group data with `.groupby()` and aggregate via `.sum()`, `.mean()`, `.count()`.

python

```
df.groupby('City')['Age'].mean()
```

Merging and Joining:

Combine multiple DataFrames with `.merge()` and `.concat()` using different types of joins.

Sorting and Ranking:

Sort dataframes by one or more columns with `.sort_values()` and rank data with `.rank()`.

Apply Functions:

Use `.apply()` to apply custom functions across rows or columns.

Input/Output:

Easy reading/writing from/to CSV, Excel, JSON, SQL databases:

python

```
df.to_csv('filename.csv')
```

```
df = pd.read_csv('filename.csv')
```

Common Data Cleaning Techniques

- Handling Missing Data: Identify missing values using functions like `isnull()`. Manage them by:
 - Removing rows or columns (`dropna()`)
 - Filling missing values (`fillna()`) with mean, median, mode, or custom logic
- Removing Duplicates: Detect duplicates with `duplicated()` and eliminate them via `drop_duplicates()`.
- Correcting Data Types: Convert columns to correct types (e.g., strings to datetime) using `astype()` or `to_datetime()`.
- Handling Outliers: Detect outliers with statistical methods (z-score, IQR) and choose to remove or transform them to reduce their impact.
- Standardizing Data: Normalize or scale numeric data using techniques like min-max scaling or z-score normalization for consistent analysis.
- Dealing with Irrelevant Data: Remove irrelevant or redundant columns that do not contribute to analysis.
- Fixing Structural Errors: Correct inconsistent naming, typos, and formatting issues to ensure uniformity.
- Encoding Categorical Variables: Convert categorical data to numerical form using one-hot encoding (`get_dummies()`) or label encoding.
- Datetime Handling: Parse and extract features from datetime fields for temporal analysis (e.g., day of week, month).

Data Preprocessing Steps

- Data Integration: Combine data from multiple sources.
- Data Transformation: Apply normalization, aggregation, or feature engineering.
- Data Reduction: Reduce data size via dimensionality reduction or sampling to improve efficiency.

Example using Pandas

python

```
import pandas as pd

df = pd.read_csv('data.csv')

# Checking missing values
print(df.isnull().sum())

# Filling missing values
df['Age'].fillna(df['Age'].mean(), inplace=True)

# Dropping duplicates
df.drop_duplicates(inplace=True)

# Converting data types
df['Date'] = pd.to_datetime(df['Date'])

# Encoding categorical variables
df = pd.get_dummies(df, columns=['Category'])

# Removing irrelevant columns
df.drop(['Unnecessary_Column'], axis=1, inplace=True)
```