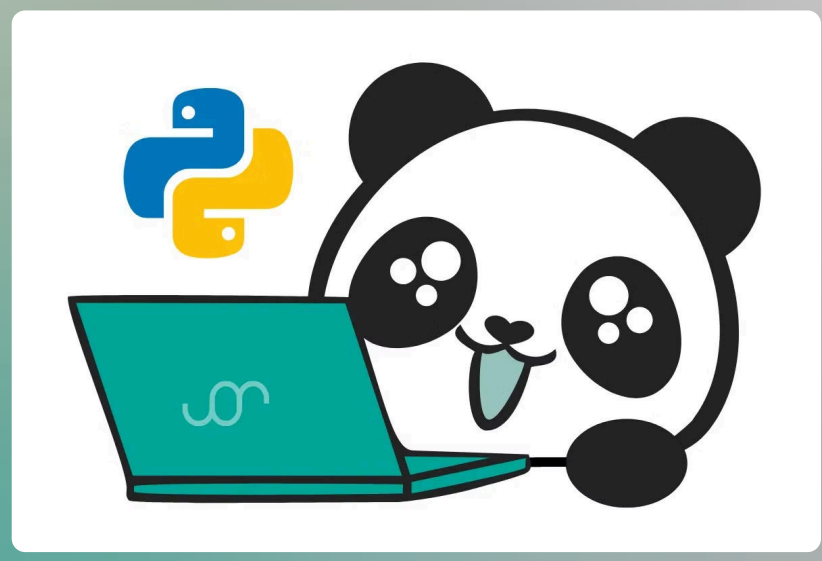
Introduction to Pandas in Data Analytics

Pandas DataFrame is an essential tool for data analysis in Python, offering a powerful and flexible tabular data structure.



1 Labeled Axes

Pandas DataFrame provides a two-dimensional, size-mutable, and potentially heterogeneous tabular data structure with labeled rows and columns.

2 Data Analysis

Commonly used alongside NumPy and Matplotlib for comprehensive data manipulation and visualization.

3 Essential for Python

Pandas DataFrame is a core component of the Python data analysis ecosystem.

```
python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

BRAINALYST
A Data Driven Company

Loading Data into a DataFrame

Methods for loading data from various sources into a DataFrame. Code Snippets:

From CSV

```
df_csv = pd.read_csv('file.csv')
```

From Excel

```
df_excel = pd.read_excel('file.xlsx', sheet_name='Sheet1')
```

From MySQL

```
import sqlalchemy engine = sqlalchemy.create_engine('mysql://username:password@localhost/dbname')
```

```
df_sql = pd.read_sql_table('table_name', engine)
```

BRAINALYST
A Data Driven Company

DataFrame and Series Objects

DataFrame: A two-dimensional table with labeled axes. Series: A one-dimensional array with labels. Index objects: Immutable array implementing an ordered, sliceable set.

Example DataFrame

```
df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})
```

Example Series

```
s = pd.Series([1, 2, 3])
```

Working with Rows and Columns

Content: Accessing and manipulating rows and columns. Selecting, adding, and deleting rows and columns.

Selecting a column

`df['A']`

Adding a new column

`df['C'] = df['A'] + df['B']`

Deleting a column

`df.drop('C', axis=1, inplace=True)`

Selecting rows

`df.loc[0] # First row df.iloc[0] # First row by position`

Indexing and Selecting Data

Content: Indexing with .loc, .iloc, and .ix. Vectorized arithmetic operations.

Using .loc

`df.loc[0:1, ['A', 'B']]`

Using .iloc

`df.iloc[0:1, 0:2]`

Vectorized operations

`df['A'] + df['B']`

Filtering and Grouping

Content:

Filtering functions and grouping by row index.

Filtering

```
filtered = df[df['A'] > 1]
```

Grouping

```
grouped = df.groupby('A').sum()
```

Merging DataFrames Title: Merging DataFrames

Merging DataFrames using pd.merge(). Types of joins: inner, outer, left, right. Code Snippets:

Creating two DataFrames

```
df1 = pd.DataFrame({'key': ['A', 'B', 'C', 'D'], 'value': [1, 2, 3, 4]})
```

```
df2 = pd.DataFrame({'key': ['B', 'D', 'E', 'F'], 'value': [5, 6, 7, 8]})
```

Inner join

```
inner_merge = pd.merge(df1, df2, on='key', how='inner')
```

Outer join

```
outer_merge = pd.merge(df1, df2, on='key', how='outer')
```

Left join

```
left_merge = pd.merge(df1, df2, on='key', how='left')
```

Right join

```
right_merge = pd.merge(df1, df2, on='key', how='right')
```

Concatenating DataFrames Title: Concatenating DataFrames

Concatenating DataFrames using pd.concat(). Concatenating along rows and columns. Code Snippets:

Concatenating along rows

```
concat_rows = pd.concat([df1, df2])
```

Concatenating along columns

```
concat_cols = pd.concat([df1, df2], axis=1)
```

Joining DataFrames Title: Joining DataFrames

Joining DataFrames using df.join(). Different types of joins: inner, outer, left, right.

Code Snippets:

Creating two DataFrames with different indexes

```
df1 = pd.DataFrame({'value1': [1, 2, 3]}, index=['A', 'B', 'C'])
df2 = pd.DataFrame({'value2': [4, 5, 6]}, index=['B', 'C', 'D'])
```

Joining DataFrames

```
joined_df = df1.join(df2, how='inner')
```

Grouping and Aggregating Data Title: Grouping and Aggregating Data

Grouping data using df.groupby(). Aggregating data using sum, mean, count, etc.

Code Snippets:

Creating a DataFrame

```
df = pd.DataFrame({ 'Category': ['A', 'B', 'A', 'B'], 'Value': [10, 20, 30, 40] })
```

Grouping by 'Category' and calculating sum

```
grouped_sum = df.groupby('Category').sum()
```

Grouping by 'Category' and calculating multiple aggregations

```
grouped_agg = df.groupby('Category').agg({'Value': ['sum', 'mean', 'count']})
```

Filtering Data Title: Filtering Data

Filtering data using conditions. Using df.query() for SQL-like queries.

Code Snippets:

Filtering with conditions

```
filtered_df = df[df['Value'] > 20]
```

Using query

```
filtered_query = df.query('Value > 20')
```

Sorting Data Title: Sorting Data

Sorting data using df.sort_values() and df.sort_index().

Code Snippets:

Sorting by values

```
sorted_values = df.sort_values(by='Value')
```

Sorting by index

```
sorted_index = df.sort_index()
```

Handling Missing Data Title: Handling Missing Data

Handling missing data with df.isna(), df.dropna(), and df.fillna().

Code Snippets:

Creating a DataFrame with missing values

```
df_missing = pd.DataFrame({ 'A': [1, np.nan, 3], 'B': [4, 5, np.nan] })
```

Checking for missing values

```
missing_values = df_missing.isna()
```

Dropping missing values

```
dropped_na = df_missing.dropna()
```

Filling missing values

```
filled_na = df_missing.fillna(0)
```

Saving and Exporting DataFrames

Content:

Saving DataFrames to various formats. Code Snippets:

To CSV

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

To Excel

```
df.to_excel('output.xlsx', sheet_name='Sheet1')
```

To Python dictionary

```
df_dict = df.to_dict()
```

To string

```
df_str = df.to_string()
```

To MySQL

```
df.to_sql('table_name', engine)
```

DataFrame Attributes and Methods

Non-indexing attributes. Utility methods. Code Snippets:

Non-indexing attributes

```
df.T
```

```
df.axes
```

```
df.dtypes
```

```
df.empty
```

```
df.ndim
```

```
df.shape
```

```
df.size
```

```
df.values
```

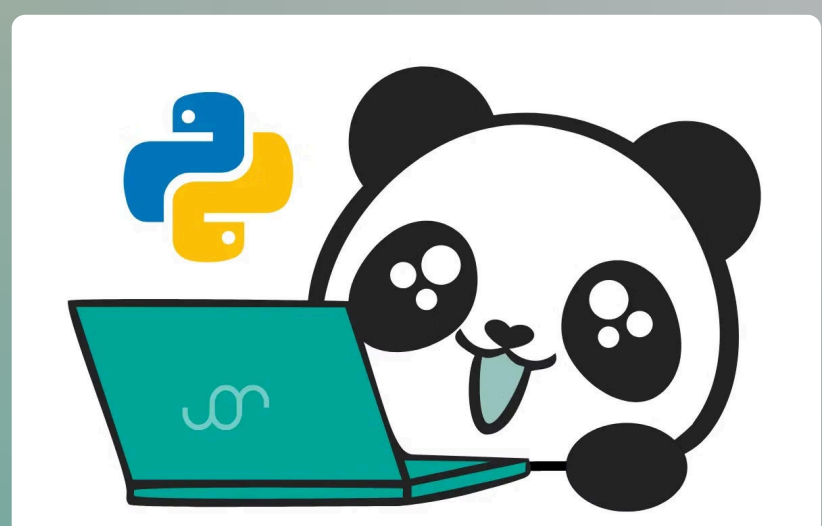
Utility methods

```
df_copy = df.copy()
```

```
df_ranked = df.rank()
```

```
df_sorted = df.sort_values(by='A')
```

```
df = df.astype({'A': 'float64'})
```



Iterating Over DataFrames Title: Iterating Over DataFrames

Methods for iterating over DataFrames.

Iterating over columns

```
for label, content in df.iteritems():
```

```
 print(label, content)
```

Iterating over rows

```
for index, row in df.iterrows():
```

```
 print(index, row)
```

Working with Dates and Times Title: Working with Dates and Time

Timestamps and Periods. Handling time zones, date ranges, and period frequencies. Code Snippets:

Timestamps

```
ts = pd.Timestamp('2023-01-01')
```

Periods

```
period = pd.Period('2023-01')
```

Date range

```
date_range = pd.date_range('2023-01-01', periods=10)
```

Period range

```
period_range = pd.period_range('2023-01', periods=10, freq='M')
```

Pivot Tables and Reshaping Data Title: Pivot Tables and Reshaping Data

Pivoting, melting, and unstacking. Code Snippets:

Pivot table

```
pivot = df.pivot_table(values='A', index='B', columns='C')
```

Melting

```
melted = pd.melt(df, id_vars=['A'], value_vars=['B', 'C'])
```

Unstacking

```
unstacked = df.unstack()
```

Slide 13: Time Series Data Title: Time Series Data

Handling time series data with DatetimeIndex and PeriodIndex. Upsampling, downsampling, and resampling. Code Snippets:

DatetimeIndex

```
dt_index = pd.DatetimeIndex(['2023-01-01', '2023-01-02'])
```

PeriodIndex

```
period_index = pd.PeriodIndex(['2023-01', '2023-02'], freq='M')
```

Resampling

```
resampled = df.resample('M').mean()
```

Additional Tips and Tricks Title: Additional Tips and Tricks

Miscellaneous tips for working with DataFrames. Code Snippets:

Value counts

```
value_counts = df['A'].value_counts()
```

Non-standard string to Timestamp conversion

```
ts = pd.to_datetime('2023-01-01 12:34:56', format='%Y-%m-%d %H:%M:%S')
```

Conclusion

Content:

- **Key Points Summary:**
  - **Introduction to Pandas DataFrame:** Understanding the basic structure and importance.
  - **Loading Data:** Methods to load data from various sources into DataFrames.
  - **DataFrame and Series Objects:** Differences and usage.
  - **Working with Rows and Columns:** Accessing, selecting, and modifying data.
  - **Indexing and Selecting Data:** Using `.loc`, `.iloc`, and vectorized operations.
  - **Saving and Exporting:** Exporting DataFrames to different formats.
  - **Attributes and Methods:** Key attributes and utility methods.
  - **Iterating Over DataFrames:** Methods to iterate through rows and columns.
  - **Dates and Times:** Handling date and time data.
  - **Pivot Tables and Reshaping:** Techniques for reshaping data.
  - **Filtering and Grouping:** Data filtering and aggregation.
  - **Time Series Data:** Managing and manipulating time series data.
  - **SQL-like Operations:** Merging, joining, concatenating, and advanced operations.
  - **Handling Missing Data:** Methods to detect and handle missing values.
- **Pandas is a powerful tool:** Pandas provides versatile and efficient methods to handle, manipulate, and analyze data, making it a cornerstone of data science and analysis in Python.