Pandas provides a comprehensive set of functions for data manipulation, aggregation, and analysis.

Data Aggregation and Statistics Functions

These functions are used to compute summary statistics and other related operations on DataFrame and Series objects.

Function	Description
`sum()`	Returns the sum of the values.
`mean()`	Returns the mean of the values.
`median()`	Returns the median of the values.
`min()`	Returns the minimum value.
`max()`	Returns the maximum value.
`std()`	Returns the standard deviation.
`var()`	Returns the variance.
`count()`	Counts non-NA cells for each column or row.
`describe()`	Generates descriptive statistics.
`quantile()`	Returns the quantile of the data.
`cumsum()`	Returns the cumulative sum.
`cumprod()`	Returns the cumulative product.
`cummax()`	Returns the cumulative maximum.
`cummin()`	Returns the cumulative minimum.

Data Manipulation Functions

These functions help in transforming data structures and contents.

Function	Description
`merge()`	Merges DataFrame objects by performing database-style join operations.
`concat()`	Concatenates pandas objects along a particular axis.
`pivot()`	Returns reshaped DataFrame organized by given index/column values.
`pivot_table()`	Creates a spreadsheet-style pivot table.
`melt()`	Unpivots a DataFrame from wide format to long format.
`cut()`	Bins values into discrete intervals.
`qcut()`	Quantile-based discretization function.

GroupBy Operations

These functions are used to split data into groups, apply a function to each group independently, and combine the results.

- groupby(): Groups data by certain criteria and allows applying functions like sum, count, etc.
- agg(): Allows multiple operations to be performed at once when combined with `groupby()`.

Window Functions

These functions are used for rolling and expanding operations.

- rolling(): Provides rolling window calculations.
- expanding(): Provides expanding transformations.

Date Functionality

Pandas provides functions to handle date and time data.

- to_datetime(): Converts argument to datetime.
- date_range(): Returns a fixed frequency DatetimeIndex.

File I/O

Pandas supports reading from and writing to different file formats.

Function	Description
`read_csv()`	Reads a comma-separated values (csv) file into DataFrame.
`to_csv()`	Writes DataFrame to a CSV file.
`read_excel()`	Reads an Excel file into a DataFrame.
`to_excel()`	Writes DataFrame to an Excel file.
`read_sql()`	Reads SQL query or database table into a DataFrame.
`to_sql()`	Writes records stored in a DataFrame to a SQL database.

Miscellaneous Functions

Function	Description
`apply()`	Applies a function along an axis of the DataFrame.
`map()`	Used for substituting each value in a Series with another value.
`applymap()`	Applies a function to a DataFrame elementwise.

Example Code

Here is an example of using some of these functions:

```
import pandas as pd

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

# Using sum and mean functions
```

```
print(df.sum())  # Sum of each column
print(df.mean())  # Mean of each column

# GroupBy operation
print(df.groupby('A').sum())

# Reading and writing to CSV
df.to_csv('data.csv')
df_read = pd.read_csv('data.csv')
```

Few more **examples**

Apply
The `apply()` function is used to apply a function along an axis of the DataFrame or on values of Series.

```
import pandas as pd
df = pd.DataFrame({
    'A': range(1, 5),
    'B': range(10, 50, 10)
})
df['A_squared'] = df['A'].apply(lambda x: x**2)
print(df)
```

Map
The `map()` function is used for substituting each value in a Series with another value.

```
s = pd.Series(['cat', 'dog', np.nan, 'rabbit'])
s = s.map({'cat': 'kitten', 'dog': 'puppy'})
print(s)
```

Applymap
The `applymap()` function is used to apply a function to a DataFrame elementwise.

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

```
df = df.applymap(lambda x: x**2)
print(df)
```

GroupBy Operations

Basic GroupBy

```
df = pd.DataFrame({
   'Key': ['A', 'B', 'A', 'B'],
   'Data': [1, 2, 3, 4]
})
print(df.groupby('Key').sum())
```

GroupBy with Aggregation

```
print(df.groupby('Key').agg({'Data': 'mean'}))
```

Custom Aggregation

```
def custom_agg(x):
   return x.max() - x.min()

print(df.groupby('Key').agg(custom_agg))
```

Window Functions

```
df = pd.DataFrame({
   'B': [0, 1, 2, np.nan, 4]
})
print(df['B'].rolling(2, min_periods=1).sum())
```

Expanding

```
print(df['B'].expanding(2).sum())
```

Date Functionality

Date Range

```
print(pd.date_range(start='1/1/2020', periods=5))
```

To Datetime

```
print(pd.to_datetime(['20200101', '20200201'], format='%Y%m%d'))
```

File I/O

Read CSV

```
# Assuming 'data.csv' exists
df = pd.read_csv('data.csv')
print(df.head())
```

To CSV

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

Data Manipulation Functions

Merge

```
df1 = pd.DataFrame({'A': ['A0', 'A1'], 'B': ['B0', 'B1']})
df2 = pd.DataFrame({'A': ['A2', 'A3'], 'B': ['B2', 'B3']})
result = pd.concat([df1, df2])
print(result)
```

Pivot

```
df = pd.DataFrame({
    'foo': ['one', 'one', 'one', 'two', 'two', 'two'],
    'bar': ['A', 'B', 'C', 'A', 'B', 'C'],
    'baz': [1, 2, 3, 4, 5, 6],
    'zoo': ['x', 'y', 'z', 'q', 'w', 't']
})
```

```
print(df.pivot(index='foo', columns='bar', values='baz'))
```

Data Aggregation and Statistics Functions

Describe

```
df = pd.DataFrame({
   'numeric1': [1, 2, 3],
   'numeric2': [4, 5, 6]
})
print(df.describe())
```

Mean

```
print(df.mean())
```