Pandas Cheat Sheet

What is Pandas?

Pandas is a powerful, open-source Python library for data manipulation and analysis. It provides data structures for efficiently storing large datasets and tools for working with them. The primary data structure in Pandas is the **DataFrame**, a two-dimensional table with labeled rows and columns, similar to a spreadsheet or SQL table.

Key Features

- **Data Structures:** DataFrame and Series (one-dimensional array).
- **Data Handling:** Reading/writing data from/to various formats (CSV, Excel, SQL, etc.).
- **Data Cleaning:** Handling missing data, data transformation.
- **Data Manipulation:** Filtering, sorting, grouping, reshaping.
- **Data Analysis:** Descriptive statistics, aggregation, merging/joining datasets.

Installation

```
pip install pandas
```

Importing Pandas

```
import pandas as pd
```

Creating DataFrames

1. From a Dictionary

```
data = {'col1': [1, 2, 3], 'col2': ['a', 'b', 'c'], 'col3': [True, False, True]}
df = pd.DataFrame(data)
print(df)
```

	col1	col2	col3
0	1	a	True
1	2	b	False
2	3	c	True

2. From a List of Lists

```
data = [[1, 'a', True], [2, 'b', False], [3, 'c', True]]
df = pd.DataFrame(data, columns=['col1', 'col2', 'col3'])
print(df)
```

	col1	col2	col3
0	1	a	True
1	2	b	False
2	3	c	True

3. From a NumPy Array

```
import numpy as np

data = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
df = pd.DataFrame(data, columns=['A', 'B', 'C'])
print(df)
```

	A	B	C
0	1	2	3
1	4	5	6
2	7	8	9

4. Reading from a CSV File

```
df = pd.read_csv('data.csv')  # Assuming you have a 'data.csv' file
```

5. Reading from an Excel File

```
df = pd.read_excel('data.xlsx', sheet_name='Sheet1')
```

Basic DataFrame Operations

Operation	Code	Description
View top rows	`df.head(n)`	Displays the first `n` rows (default is 5).
View bottom rows	`df.tail(n)`	Displays the last `n` rows (default is 5).

Operation	Code	Description
DataFrame info	`df.info()`	Provides a summary of the DataFrame, including data types and non-null values.
Descriptive stats	`df.describe()`	Generates descriptive statistics (e.g., mean, std, min, max).
Shape	`df.shape`	Returns the dimensions (rows, columns) of the DataFrame.
Columns	`df.columns`	Returns the column labels of the DataFrame.
Index	`df.index`	Returns the index (row labels) of the DataFrame.
Data types	`df.dtypes`	Returns the data types of each column.

Pandas Cheat Sheet: Data Selection, Indexing, and Slicing

Selecting Columns

1. Single Column

```
# Returns a Series
col2_series = df['col2']
print(col2_series)
```

```
0    a
1    b
2    c
Name: col2, dtype: object
```

2. Multiple Columns

```
# Returns a DataFrame
subset_df = df[['col1', 'col3']]
print(subset_df)
```

	col1	col3
0	1	True
1	2	False
2	3	True

Selecting Rows

1. By Label (loc)

```
# Select row with index label 0
row_0 = df.loc[0]
print(row_0)
```

```
col1       1
col2       a
col3    True
Name: 0, dtype: object
```

```
# Select rows with index labels 0 and 2
rows_0_2 = df.loc[[0, 2]]
print(rows_0_2)
```

	col1	col2	col3
0	1	a	True
2	3	c	True

2. By Position (iloc)

```
# Select the first row (index position 0)
first_row = df.iloc[0]
print(first_row)
```

```
col1       1
col2       a
col3    True
Name: 0, dtype: object
```

```
# Select the first two rows
first_two_rows = df.iloc[:2]
print(first_two_rows)
```

	col1	col2	col3

	col1	col2	col3
0	1	a	True
1	2	b	False

Selecting Rows and Columns

1. Using loc

```
# Select row with index label 1, and columns 'col2' and 'col3'
subset = df.loc[1, ['col2', 'col3']]
print(subset)
```

```
col2        b
col3    False
Name: 1, dtype: object
```

2. Using iloc

```
# Select the element at the first row (index 0) and second column (index 1)
element = df.iloc[0, 1]
print(element)
```

```
a
```

Boolean Indexing (Filtering)

```
# Select rows where 'col1' is greater than 1
filtered_df = df[df['col1'] > 1]
print(filtered_df)
```

	col1	col2	col3
1	2	b	False
2	3	c	True

```
# Select rows where 'col3' is True and 'col1' is not equal to 1
filtered_df = df[(df['col3'] == True) & (df['col1'] != 1)]
print(filtered_df)
```

	col1	col2	col3
2	3	c	True

Setting Values

1. Using loc

```
# Set the value of 'col2' at index label 0 to 'z'
df.loc[0, 'col2'] = 'z'
print(df)
```

	col1	col2	col3
0	1	z	True
1	2	b	False
2	3	c	True

2. Using Boolean Indexing

```
# Set the value of 'col1' to 0 where 'col3' is False
df.loc[df['col3'] == False, 'col1'] = 0
print(df)
```

	col1	col2	col3
0	1	z	True
1	0	b	False
2	3	c	True

Pandas Cheat Sheet: Data Cleaning and Transformation

Handling Missing Data

1. Identifying Missing Data

```
# Create a DataFrame with missing values (NaN)
data = {'A': [1, 2, np.nan], 'B': [4, np.nan, 6], 'C': [7, 8, 9]}
df = pd.DataFrame(data)
print(df)
```

	A	B	C
0	1	4	7
1	2	NaN	8
2	NaN	6	9

```
# Check for missing values (returns a DataFrame of booleans)
print(df.isnull())
```

	A	B	C
0	False	False	False
1	False	True	False
2	True	False	False

```
# Count missing values per column
print(df.isnull().sum())
```

```
A    1
B    1
C    0
dtype: int64
```

2. Dropping Missing Data

```
# Drop rows with any missing values
df_dropna = df.dropna()
print(df_dropna)
```

	A	B	C
0	1	4	7

```
# Drop columns with any missing values
df_dropna_cols = df.dropna(axis=1)
print(df_dropna_cols)
```

	C
0	7
1	8
2	9

3. Filling Missing Data

```
# Fill missing values with a specific value (e.g., 0)
df_fillna = df.fillna(0)
print(df_fillna)
```

	A	B	C
0	1	4	7
1	2	0	8
2	0	6	9

```
# Fill missing values with the mean of each column
df_fillna_mean = df.fillna(df.mean())
print(df_fillna_mean)
```

	A	B	C
0	1	4	7
1	2	5	8
2	1.5	6	9

Data Transformation

1. Applying Functions

```
# Apply a function to each element in a column
df['A_squared'] = df['A'].apply(lambda x: x**2)
print(df)
```

	A	B	C	A_squared
0	1	4	7	1
1	2	NaN	8	4

	A	B	C	A_squared
2	NaN	6	9	NaN

```
# Apply a function to the entire DataFrame
def add_one(x):
    return x + 1
df = df.apply(add_one)
print(df)
```

	A	B	C	A_squared
0	2	5	8	2
1	3	NaN	9	5
2	NaN	7	10	NaN

2. Adding/Removing Columns

```
# Add a new column
df['D'] = [10, 11, 12]
print(df)
```

	A	B	C	A_squared	D
0	2	5	8	2	10
1	3	NaN	9	5	11
2	NaN	7	10	NaN	12

```
# Remove a column
df = df.drop('A_squared', axis=1)
print(df)
```

	A	B	C	D
0	2	5	8	10
1	3	NaN	9	11
2	NaN	7	10	12

3. Renaming Columns

```
# Rename columns
df = df.rename(columns={'A': 'col_A', 'B': 'col_B'})
print(df)
```

	col_A	col_B	C	D
0	2	5	8	10
1	3	NaN	9	11
2	NaN	7	10	12

4. Changing Data Types

```
# Convert a column to a different data type
df['col_A'] = df['col_A'].astype('Int64')  # Convert to nullable integer type
print(df.dtypes)
```

```
col_A      Int64
col_B    float64
C          int64
D          int64
dtype: object
```

Pandas Cheat Sheet: Data Aggregation and Grouping

Descriptive Statistics

Function	Description
`df.mean()`	Calculates the mean of each column.
`df.median()`	Calculates the median of each column.
`df.min()`	Finds the minimum value in each column.
`df.max()`	Finds the maximum value in each column.
`df.std()`	Calculates the standard deviation of each column.
`df.sum()`	Calculates the sum of each column.
`df.count()`	Counts the number of non-missing values in each column.
`df.quantile(q)`	Calculates the q-th quantile (e.g., 0.25 for the first quartile, 0.5 for the median).

Grouping Data

1. groupby()

```
# Create a sample DataFrame
data = {'group': ['A', 'A', 'B', 'B', 'C'],
        'value1': [10, 12, 15, 18, 20],
        'value2': [5, 7, 9, 11, 13]}
df = pd.DataFrame(data)
print(df)
```

	group	value1	value2
0	A	10	5
1	A	12	7
2	B	15	9
3	B	18	11
4	C	20	13

```
# Group data by the 'group' column
grouped = df.groupby('group')
```

2. Aggregation

```
# Calculate the mean of each column for each group
group_means = grouped.mean()
print(group_means)
```

	value1	value2
group		
A	11	6
B	16.5	10
C	20	13

```
# Calculate the sum of 'value1' and the maximum of 'value2' for each group
group_agg = grouped.agg({'value1': 'sum', 'value2': 'max'})
print(group_agg)
```

	value1	value2
group		
A	22	7
B	33	11
C	20	13

```
# Apply multiple aggregation functions to each column
group_agg_multi = grouped.agg(['mean', 'std', 'min', 'max'])
print(group_agg_multi)
```

	value1	value2
	mean	std
group		
A	11	1.414214
B	16.5	2.12132
C	20	NaN

Pivot Tables

```
# Create a pivot table to show the mean of 'value1' for each group, with 'group'
as index and 'value2' categories as columns
pivot_table = df.pivot_table(values='value1', index='group', columns='value2',
aggfunc='mean')
print(pivot_table)
```

value2	5	7	9	11	13
group					
A	10	12	NaN	NaN	NaN
B	NaN	NaN	15	18	NaN
C	NaN	NaN	NaN	NaN	20

Pandas Cheat Sheet: Reshaping and Combining DataFrames

Reshaping Data

1. `melt()` (Unpivot)

Converts a "wide" DataFrame to a "long" format.

```
# Create a wide DataFrame
data = {'Name': ['Alice', 'Bob'],
        'Math': [80, 90],
        'Science': [85, 92],
        'English': [78, 88]}
df_wide = pd.DataFrame(data)
print(df_wide)
```

	Name	Math	Science	English
0	Alice	80	85	78
1	Bob	90	92	88

```
# Melt the DataFrame to make it long
df_long = df_wide.melt(id_vars=['Name'], var_name='Subject', value_name='Score')
print(df_long)
```

	Name	Subject	Score
0	Alice	Math	80
1	Bob	Math	90
2	Alice	Science	85
3	Bob	Science	92
4	Alice	English	78
5	Bob	English	88

2. `pivot()` (Pivot)

Converts a "long" DataFrame to a "wide" format (opposite of `melt()`).

```
# Pivot the long DataFrame back to wide format
df_wide_again = df_long.pivot(index='Name', columns='Subject', values='Score')
print(df_wide_again)
```

Subject	English	Math	Science
Name			

Subject	English	Math	Science
Alice	78	80	85
Bob	88	90	92

3. `stack()` and `unstack()`

`stack()`: Reshapes a DataFrame by stacking the specified level(s) from columns to index, making the DataFrame taller. `unstack()`: Performs the inverse operation of stack, moving levels from the index to the columns, making the DataFrame wider.

```
# Assuming df_wide_again is already stacked
df_stacked = df_wide_again.stack()
print("Stacked DataFrame:\n", df_stacked)

# Unstacking the DataFrame
df_unstacked = df_stacked.unstack()
print("\nUnstacked DataFrame:\n", df_unstacked)
```

```
Stacked DataFrame:
 Name   Subject
Alice  English    78.0
       Math       80.0
       Science    85.0
Bob    English    88.0
       Math       90.0
       Science    92.0
dtype: float64

Unstacked DataFrame:
 Subject  English  Math  Science
Name
Alice       78.0  80.0     85.0
Bob         88.0  90.0     92.0
```

Combining DataFrames

1. `concat()`

Concatenates DataFrames along rows (vertically) or columns (horizontally).

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

	A	B
0	A0	B0
1	A1	B1

	A	B
0	A2	B2
1	A3	B3

```
# Concatenate vertically (along rows)
df_concat_rows = pd.concat([df1, df2])
print(df_concat_rows)
```

	A	B
0	A0	B0
1	A1	B1
0	A2	B2
1	A3	B3

```
# Concatenate horizontally (along columns)
df_concat_cols = pd.concat([df1, df2], axis=1)
print(df_concat_cols)
```

	A	B	A	B
0	A0	B0	A2	B2
1	A1	B1	A3	B3

2. `merge()`

Merges DataFrames based on common columns (like SQL joins).

```
# Create two DataFrames with a common column
left = pd.DataFrame({'key': ['K0', 'K1', 'K2'], 'A': ['A0', 'A1', 'A2']})
right = pd.DataFrame({'key': ['K0', 'K1', 'K3'], 'B': ['B0', 'B1', 'B3']})
print(left)
print(right)
```

	key	A

	key	A
0	K0	A0
1	K1	A1
2	K2	A2

	key	B
0	K0	B0
1	K1	B1
2	K3	B3

```
# Inner merge (only common keys)
df_inner = pd.merge(left, right, on='key', how='inner')
print(df_inner)
```

	key	A	B
0	K0	A0	B0
1	K1	A1	B1

```
# Left merge (all keys from left DataFrame)
df_left = pd.merge(left, right, on='key', how='left')
print(df_left)
```

	key	A	B
0	K0	A0	B0
1	K1	A1	B1
2	K2	A2	NaN

```
# Right merge (all keys from right DataFrame)
df_right = pd.merge(left, right, on='key', how='right')
print(df_right)
```

	key	A	B
0	K0	A0	B0
1	K1	A1	B1
2	K3	NaN	B3

```
# Outer merge (all keys from both DataFrames)
df_outer = pd.merge(left, right, on='key', how='outer')
print(df_outer)
```

	key	A	B
0	K0	A0	B0
1	K1	A1	B1
2	K2	A2	NaN
3	K3	NaN	B3