Methods

.copy()
.drop()
.where()
.query()
.nlargest()
.nsmallest()

.copy()

The copy() method is used to create a deep copy of a DataFrame. This is especially important when slicing or selecting a part of a DataFrame.

Why Use copy()?

When you assign a column or a slice of a DataFrame to a variable, it does not create a copy by default—it creates a reference.

Any changes made to the variable will also affect the original DataFrame.

.copy()

```
col = df['Rating']
col[0] = 100  # Changes the original DataFrame!
print(df)
```

```
col_copy = df['Rating'].copy()
col_copy[0] = 100  # Does not affect the original DataFrame
print(df)
```

.drop()

The drop() method is used to remove rows or columns from a DataFrame

```
DataFrame.drop(labels, axis=0, inplace=False)
```

Parameters:

- **labels:** The row or column labels to drop.
- **axis**: 0 for rows, 1 for columns.
- **inplace:** If True, modifies the original DataFrame.

.drop()

Dropping a row

```
#drop the row labled 'Almond Delight', not  permanently
df.drop('Almond Delight')
```

Dropping a column

```
# Drop the 'Rating' column
df_dropped = df.drop('Rating', axis=1)
```

.where()

The where() method is used to filter data, retaining values that satisfy a condition and replacing others with NaN.

```
#recreate evertihng
mask_k = cereal["mfr"]=='K'
cereal[mask_k]
cereal.where(mask_k)
```

name	mfr	type	calories	protein	fat	sodium	fiber	carbo	sugars	potass	vitamins	shelf	weight	cups	rating
100% Bran	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
100% Natural Bran	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
All-Bran	K	C	70.0	4.0	1.0	260.0	9.0	7.0	5.0	320.0	25.0	3.0	1.0	0.33	59.425505
All-Bran with Extra Fiber	K	C	50.0	4.0	0.0	140.0	14.0	8.0	0.0	330.0	25.0	3.0	1.0	0.50	93.704912
Almond Delight	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

Condition NOT satisfied

Condition satisfied

.query()

The query() method is used to filter rows based on an expression

```
DataFrame.query(expr)
```

Advantages of query() Over Regular Filtering:

1. Readability:

query() uses string-based expressions that resemble SQL-like syntax, making it easier to understand for those familiar with SQL.

```
filtered = df.query('Rating > 45')
```

2. Complex Conditions:Writing complex conditions involving multiple columns is cleaner with query().

```
df.query('Rating > 45 and Sugars < 10')
```

.query()

String within a string

```
df.query("last_name == 'Hovy'")
```

Example with a string

```
df[df["last_name"]=='Hovy']
```

The same result, different syntax

	first_name	last_name	salary	start_date	gender	remote	team
351	Kory	Hovy	426422	2020-10-05	NaN	True	marketing

Comparing loc and iloc

Feature	where()	query()
Purpose	Filter data conditionally, replacing others with NaN.	Filter rows using a string-based expression.
Condition Type	Works directly with Boolean masks.	Works with string expressions.
Output	Retains DataFrame structure, replacing unmatched rows.	Returns only rows matching the condition.
Example	df.where(df['Rating'] > 45)	df.query('Rating > 45')
Performance	Generally slower for complex conditions.	Faster due to optimized evaluation.

.nlargest() and nsmallest()

These methods are used to get the n largest or smallest values in a specific column.

```
DataFrame.nlargest(n, columns)
DataFrame.nsmallest(n, columns)
```

Parameters:

n: Number of rows to return.

columns: Column to consider for ranking.

```
top_rated = df.nlargest(2, 'Rating')
```

```
lowest_rated = df.nsmallest(2, 'Rating')
```