Suppose df is a DataFrame; s is a Series. import pandas as pd

Function	**Description**
df.shape	Returns a tuple containing the number of rows and columns, in that order
df.index	Returns the index (row labels) of df as an Index objec
df[col]	Returns the column labeled col from df as a Series
df[[col1, col2]]	Returns a DataFrame containing the columns labeled col1 and col2
s.astype(dtype)	Returns a Series casted to the specified type dtype
s.loc[rows] / df.loc[rows, cols]	Returns a Series/DataFrame with rows (and columns) selected by their index values
s.iloc[rows] / df.iloc[rows, cols]	Returns a Series/DataFrame with rows (and columns) selected by their positions
s.isnull() / df.isnull()	Returns boolean Series/DataFrame identifying missing values
s.isin(values) / df.isin(values)	Returns a Series/DataFrame of booleans indicating if each element is in values.
df.drop(labels, axis)	Returns a DataFrame without the rows or columns named labels along axis (either 0 or 1)
df.rename(index=None, columns=None)	Returns a DataFrame with renamed columns from a dictionary index and/or columns
df.sort_values(by, ascending=True)	Returns a DataFrame where rows are sorted by the values in columns by
s.sort_values(ascending=True)	Returns a sorted Series
s.unique()	Returns a NumPy array of the unique values
s.value_counts()	Returns the number of times each unique value appears in a Series
pd.merge(left, right, how='inner', left_on=col1, right_on=col2)	Returns a DataFrame joining left and right on columns labeled col1 and col2; the join is of type inner
left.merge(right, left_on=col1, right_on=col2)	Returns a DataFrame joining left and right on columns labeled col1 and col2
pd.melt(frame, id_vars=None, value_vars=None,var_name=None, value_name='value')	Returns a DataFrame that unpivots a DataFrame from wide to long format, increasing the number of rows and decreasing the number of columns
df.pivot_table(values=None, index=None, columns=None, aggfunc='mean', fill_value=None)	Returns a DataFrame pivot table where columns are unique values from columns (column name or list), and rows are unique values from index (column name or list); cells are collected values using aggfunc. If values is not provided, cells are collected for each remaining column with multi-level column indexing
df.set_index(col)	Returns a DataFrame that uses the values in the column labeled col as the row index
df.reset_index()	Returns a DataFrame that has row index 0, 1, etc., and adds the current index as a column

Let grouped = df.groupby(by) where by can be a column label or a list of labels

Function	Description
grouped.count()	Return a DataFrame containing the size of each group, excluding missing values
grouped.size()	Return a Series containing size of each group, including missing values
grouped.mean()/.std()/.min()/.max()	Return a Series/DataFrame containing mean/std/min/max of each group for each column, excluding missing values

grouped.filter(f)
grouped.agg(f)
	Filters or aggregates using the given function f

Function	Description
s.str.lower()/s.str.upper()	Returns a Series of lowercase/uppercase versions of each string
s.str.replace(pat, repl, regex=False)	Returns a Series that replaces occurences of substrings matching pat with string repl. When regex=False, pat is treated as a literal string; when regex=True, pat is treated as a RegEx pattern.
s.str.contains(pat)	Returns a boolean Series indicating if a substring matching the regex pat is contained in each string
s.str.extract(pat)	Returns a DataFrame of the first subsequence of each string that matches the regex pat. If pat contains one group, then only the substring matching the group is extracted
s.str.split(pat=" ")	Splits the strings in s at the delimiter pat (defaults to a whitespace). Returns a Series of lists, where each list contains strings of the characters before and after the split.
s.str[i]	Extracts the character at position i from each string in s. Returns a Series with the selected characters.

Visualization

Function	Description
ggplot(data=None, mapping=None)	Creates a new ggplot object. It is used as the foundation for building plots layer by layer. 'data' is the dataset to be plotted, and 'mapping' defines the aesthetic (aes) mappings.
geom_bar(mapping=None, data=None, stat='count', position='stack')	Adds a bar chart layer to the plot. mapping defines aesthetic mappings, data overrides the plot data, stat specifies the statistical transformation, and position adjusts the position of overlapping objects. stat=”identity” to plot y in mapping.
geom_boxplot(mapping=None, data=None, stat='boxplot', position='dodge')	Adds a box plot layer to the plot. Arguments function similarly to other geoms.
geom_density(mapping=None, data=None, stat='density', position='identity')	Adds a density plot layer to the plot. Arguments function similarly to other geoms.
geom_freqpoly(mapping=None, data=None, stat='bin', position='identity')	Adds a frequency polygon layer to the plot. Arguments work as in geom_histogram().

geom_histogram(mapping=None, data=None, stat='bin', position='stack')	Adds a histogram layer to the plot. Arguments function similarly to other geoms.
geom_line(mapping=None, data=None, stat='identity', position='identity')	Adds a line plot layer to the plot. Arguments function similarly to other geoms.
geom_point(mapping=None, data=None, stat='identity', position='identity')	Adds a scatter plot layer to the plot. Arguments function as in previous geoms.
geom_tile(mapping=None, data=None, stat='identity', position='identity')	Adds a tile plot layer to the plot. Arguments function similarly to other geoms.