Pandas Functions

- **read_csv():** Read a CSV file into a DataFrame.

"df = pd.read_csv('data.csv')"

- **read_excel():** Read an Excel file into a DataFrame.

"df = pd.read_excel('data.xlsx')"

- **to_csv():** Write DataFrame to a CSV file.

"df.to_csv('data_out.csv')"

- **to_excel():** Write DataFrame to an Excel file.

"df.to_excel('data_out.xlsx')"

- **head():** View the first n rows of the DataFrame.

"df.head()"

- **tail():** View the last n rows of the DataFrame.

"df.tail()"

- **info():** Get concise summary of the DataFrame.

"df.info()"

- **describe():** Generate descriptive statistics.

"df.describe()"

- **shape**: Get the dimensions of the DataFrame.

"df.shape"

- **columns**: Get the column names of the DataFrame.

"df.columns"

- **index**: Get the index (row labels) of the DataFrame.

"df.index"

- **values**: Get the data as a 2D array.

"df.values"

- **dtypes**: Get the data types of each column.

"df.dtypes"

- **astype():** Convert the data type of a column.

"df['column'].astype('int')"

- **fillna():** Fill missing values in DataFrame.

"df.fillna(0)"

- **dropna():** Remove missing values from DataFrame.

"df.dropna()"

- **drop():** Drop specified labels from rows or columns.

"df.drop(['column1', 'column2'], axis=1)"

- **iloc[]:** Access DataFrame by integer*location.

"df.iloc[0]"

- **loc[]:** Access DataFrame by label*location.

"df.loc['label']"

- **groupby():** Group DataFrame using a mapper or by a series of columns.

"df.groupby('column')"

- **agg():** Apply aggregation functions to grouped data.

"df.groupby('column').agg({'column2': 'sum'})"

- **merge():** Merge DataFrame or Series with a database*style join.

"pd.merge(df1, df2, on='key_column')"

- **concat():** Concatenate DataFrame objects along a particular axis.

"pd.concat([df1, df2])"

- **apply():** Apply a function along an axis of the DataFrame.

"df.apply(np.sqrt)"

- **applymap():** Apply a function element*wise in the DataFrame.

"df.applymap(lambda x: x*2)"

- **map():** Apply a function to every element in a column.

"df['column'].map(lambda x: x*2)"

- **pivot_table():** Create a spreadsheet*style pivot table.

"pd.pivot_table(df, values='column1', index='column2', aggfunc=np.sum)"

- **melt():** Unpivot a DataFrame from wide to long format.

"pd.melt(df, id_vars=['column1'], value_vars=['column2', 'column3'])"

- **sort_values():** Sort DataFrame by column values.

"df.sort_values(by='column')"

- **sort_index():**Sort DataFrame by index labels.

"df.sort_index()"

- **set_index():** Set the DataFrame index (row labels) using existing columns.

"df.set_index('column')"

- **reset_index():** Reset the DataFrame index.

"df.reset_index()"

- **value_counts():** Count unique values in a column.

"df['column'].value_counts()"

- **unique():** Get unique values in a column.

"df['column'].unique()"

- **nunique():** Count distinct observations in a column.

"df['column'].nunique()"

- **drop_duplicates():():** Remove duplicate rows from DataFrame.

"df.drop_duplicates()"

- **isin():** Filter DataFrame rows based on a list of values.

"df[df['column'].isin(['value1', 'value2'])]"

- **query():** Query the DataFrame with a boolean expression.

"df.query('column > 0')"

- **rename():** Rename columns or index labels.

"df.rename(columns={'old_name': 'new_name'})"

- **cut():** Bin values into discrete intervals.

"pd.cut(df['column'], bins=3)"

- **qcut():** Bin values into quantiles.

"pd.qcut(df['column'], q=4)"

- **rolling():** Provide rolling window calculations.

"df['column'].rolling(window=3).mean()"

- **shift():** Shift index by desired number of periods.

"df['column'].shift(periods=1)"

- **str**: String methods for Series.

"df['column'].str.lower()"

- **datetime**: Convert column to datetime dtype.

"pd.to_datetime(df['date_column'])"

- **pd.to_numeric():** Convert argument to a numeric type.

"pd.to_numeric(df['numeric_column'])"

- **get_dummies():** Convert categorical variables into dummy/indicator variables.

"pd.get_dummies(df['category_column'])"

- str.contains(): Check if pattern or regex is contained within a string of a Series.

"df['column'].str.contains('pattern')"

- **str.replace():** Replace occurrences of pattern/regex in the Series/Index with some other string.

"df['column'].str.replace('old_string', 'new_string')"

- **str.extract():** Extract capture groups in the regex pat as columns in a DataFrame.

"df['column'].str.extract('(\d+)')"

- **str.split():** Split strings around given separator/delimiter.

"df['column'].str.split(',')"

- **str.strip():** Remove leading and trailing characters.

"df['column'].str.strip()"

- **str.join():** Join lists contained as elements in the Series/Index with passed delimiter.

"','.join(df['column'])"

- **str.isnumeric():** Check whether all characters in each string are numeric.

"df['column'].str.isnumeric()"

- **str.isalpha():** Check whether all characters in each string are alphabetic.

"df['column'].str.isalpha()"

- **str.isdigit():** Check whether all characters in each string are digits.

"df['column'].str.isdigit()"

- **str.startswith()**: Check whether each string starts with a specified substring.

"df['column'].str.startswith('prefix')"

- **str.endswith():** Check whether each string ends with a specified suffix.

"df['column'].str.endswith('suffix')"

- **str.find():** Return lowest indexes in each string in the Series/Index where the substring is fully contained between [start:end].

"df['column'].str.find('substring')"

- **str.len():** Return the length of the string.

"df['column'].str.len()"

- **str.capitalize():** Convert strings in the Series/Index to be capitalized.

"df['column'].str.capitalize()"

- **str.lower():** Convert strings in the Series/Index to lowercase.

"df['column'].str.lower()"

- **str.upper():** Convert strings in the Series/Index to uppercase.

"df['column'].str.upper()"

- **str.title():** Convert strings in the Series/Index to titlecase.

"df['column'].str.title()"

- **str.swapcase():** Convert strings in the Series/Index to swapcase.

"df['column'].str.swapcase()"

- **str.isalnum():** Check whether all characters in each string are alphanumeric.

"df['column'].str.isalnum()"

- **str.isdecimal():** Check whether all characters in each string are decimal.

"df['column'].str.isdecimal()"

- **str.islower():** Check whether all characters in each string are lowercase.

"df['column'].str.islower()"

- **str.isupper():** Check whether all characters in each string are uppercase.

"df['column'].str.isupper()"

- **str.isspace():** Check whether all characters in each string are whitespace.

"df['column'].str.isspace()"

- **str.count():** Count occurrences of pattern in each string of the Series/Index.

"df['column'].str.count('pattern')"

- **str.pad():** Filling the strings in the Series/Index to a minimum width.

"df['column'].str.pad(width=10)"

- **str.lstrip():** Stripping whitespace from the beginning of each string in the Series/Index.

"df['column'].str.lstrip()"

- **str.rstrip():** Stripping whitespace from the end of each string in the Series/Index.

"df['column'].str.rstrip()"

- **str.rjust()**: Filling the right side of strings in the Series/Index with an additional character.

"df['column'].str.rjust(width=10, fillchar='0')"

- **str.ljust():** Filling left side of strings in the Series/Index with an additional character.

"df['column'].str.ljust(width=10, fillchar='0')"

- **str.zfill():** Pad strings in the Series/Index by prepending ‘0’ characters.

"df['column'].str.zfill(width=10)"

- **str.partition():** Split the strings in the Series/Index around the first occurrence of sep.

"df['column'].str.partition(sep='delimiter')"

- **str.rpartition():** Split the strings in the Series/Index around the last occurrence of sep.

"df['column'].str.rpartition(sep='delimiter')"

- **str.splitlines():** Split the strings in the Series/Index at the line breaks.

"df['column'].str.splitlines()"

- **str.expandtabs():** Expand tabs in each string of the Series/Index.

"df['column'].str.expandtabs(tabsize=4)"

- **str.swapaxes():** Swap axes of the Series/Index objects.

"df['column'].str.swapaxes(axis1=0, axis2=1)"