🐼 PANDAS COMPLETE FUNCTION + SYNTAX CHEATSHEET

IMPORT & BASIC SETUP

```
import pandas as pd
```

CREATE DATA STRUCTURES

```
# Series
s = pd.Series([1,2,3,4], name='col1')
# DataFrame
df = pd.DataFrame({'A':[1,2,3],'B':[4,5,6]})
# From lists / dict / numpy
pd.DataFrame(list_of_lists, columns=['A','B'])
pd.DataFrame.from_dict(dict_data)
pd.DataFrame.from_records(record_list)
pd.Series(np_array)
```

READ & WRITE DATA

```
pd.read_csv('file.csv')
pd.read_excel('file.xlsx', sheet_name=0)
pd.read_json('file.json')
pd.read_sql('SELECT * FROM table', conn)
pd.read_html('https://...')
pd.read_parquet('file.parquet')
pd.read_clipboard()
df.to_csv('out.csv', index=False)
df.to_excel('out.xlsx', index=False)
df.to_json('out.json')
df.to_sql('table', conn, if_exists='replace')
df.to_parquet('out.parquet')
```

VIEW & INFO

```
df.head()          # first 5 rows
df.tail(10)        # last 10 rows
df.info()          # column info
df.shape           # (rows, cols)
df.columns         # column names
df.dtypes          # datatypes
df.describe()      # stats summary
df.memory_usage()
df.sample(5)
```

INDEXING & SELECTION

```
df['A']              # column
df[['A','B']]        # multiple cols
```

```
df.loc[row_label, col_label]
df.loc[0:5, ['A','B']]
df.iloc[0:5, 0:2]
df.at[0,'A']
df.iat[0,0]
df.query('A > 10')
df.filter(like='A')
df.select_dtypes(include='number')
```

BASIC OPERATIONS

```
df['new'] = df['A'] + df['B']
df.rename(columns={'A':'colA'}, inplace=True)
df.drop('A', axis=1, inplace=True)
df.drop([0,1], axis=0)
df.insert(2, 'C', df['A']*2)
df.sort_values('A', ascending=False)
df.sort_index()
df.reset_index(drop=True)
df.set_index('A', inplace=True)
df.reindex(range(10))
```

HANDLING MISSING DATA

```
df.isna()
df.notna()
df.dropna()
df.dropna(subset=['A'])
df.fillna(0)
df.fillna(df.mean())
df.interpolate()
df.replace('?', np.nan)
df.replace({np.nan:0})
df.ffill()
df.bfill()
```

FILTERING & CONDITIONS

```
df[df['A'] > 5]
df[(df['A']>5) & (df['B']<10)]
df.query('A == 10 or B < 5')
df['A'].between(5,10)
```

STRING OPERATIONS (.str)

```
df['A'].str.lower()
df['A'].str.upper()
df['A'].str.contains('abc')
df['A'].str.replace('old','new')
df['A'].str.strip()
df['A'].str.len()
df['A'].str.split(',')
```

```
df['A'].str.cat(df['B'], sep='-')
```

DATE & TIME (.dt)

```
pd.to_datetime(df['date'])
df['date'].dt.year
df['date'].dt.month
df['date'].dt.day
df['date'].dt.hour
df['date'].dt.strftime('%Y-%m-%d')
df['date'] = pd.to_datetime(df['date'])
```

GROUPING & AGGREGATION

```
df.groupby('A').sum()
df.groupby('A')['B'].mean()
df.groupby(['A','C']).agg({'B':['sum','mean']})
df['A'].value_counts()
df.pivot_table(values='B', index='A', columns='C', aggfunc='mean')
df.crosstab(df['A'], df['B'])
```

MERGE / JOIN / CONCAT

```
pd.concat([df1, df2], axis=0)
pd.concat([df1, df2], axis=1)
pd.merge(df1, df2, on='key')
pd.merge(df1, df2, how='left', on='id')
pd.merge(df1, df2, how='right', on='id')
pd.merge(df1, df2, how='outer', on='id')
df.join(df2, lsuffix='_L', rsuffix='_R')
```

APPLY / MAP / LAMBDA

```
df['A'].map(lambda x: x*2)
df['B'].apply(len)
df.apply(np.sum, axis=0)
df.apply(lambda row: row['A']+row['B'], axis=1)
```

DUPLICATES & UNIQUE

```
df.duplicated()
df.drop_duplicates()
df['A'].unique()
df['A'].nunique()
```

NUMERICAL & STATS

```
df.sum()
df.mean()
df.median()
df.std()
df.var()
```

```
df.min()
df.max()
df.corr()
df.cov()
df.rank()
df.clip(lower=0)
```

WINDOW FUNCTIONS

```
df['A'].rolling(3).mean()
df['A'].expanding().sum()
df['A'].ewm(span=3).mean()
```

MULTIINDEX

```
df.set_index(['A','B'])
df.unstack()
df.stack()
df.swaplevel()
df.sort_index(level=0)
```

CATEGORY & TYPE CONVERSION

```
df['A'] = df['A'].astype('category')
df['A'] = df['A'].astype('int64')
df.convert_dtypes()
df.select_dtypes(include='object')
```

JSON / DICT CONVERSION

```
df.to_dict()
df.to_dict('records')
pd.json_normalize(json_data)
```

STYLING (for Jupyter)

```
df.style.highlight_max()
df.style.format({'A': '{:.2f}'})
df.style.bar(subset=['A'])
```

ADVANCED OPS

```
df.eval('C = A + B')
df.pipe(func)
df.explode('list_column')
df.melt(id_vars=['A'], value_vars=['B','C'])
df.assign(new=lambda x: x.A + x.B)
df.query('A > 10')
```

PLOT (built-in)

```
df.plot()
```

```
df['A'].plot(kind='hist')
df.plot(kind='bar', x='A', y='B')
df.plot.scatter(x='A', y='B')
df.boxplot()
```

EXPORT SUMMARY

```
df.info()
df.describe(include='all')
df.value_counts()
```

FILE HANDLING SHORTCUTS

```
pd.read_pickle('file.pkl')
df.to_pickle('file.pkl')
```

PERFORMANCE

```
df.memory_usage(deep=True)
df.optimize()   # (Pandas 2.x extension / use third-party)
```

🧠 PANDAS ADVANCED & RARELY USED (COMPLETE EXTENSION SET)

INDEX OPERATIONS

```
df.index.name = 'id'
df.columns.name = 'features'
df.rename_axis('index_name')
df.index.to_list()
df.index.get_level_values(0)
df.index.is_monotonic_increasing
df.index.is_unique
df.reorder_levels(['B','A'], axis=0)
```

SORTING ADVANCED

```
df.sort_values(by=['A','B'], ascending=[True, False])
df.sort_index(axis=1)
df.nlargest(5, 'A')
df.nsmallest(5, 'B')
```

ADVANCED GROUPBY OPS

```
df.groupby('A').transform('mean')
df.groupby('A').apply(lambda x: x.head(1))
df.groupby('A', group_keys=False).apply(lambda x: x.nlargest(1, 'B'))
df.groupby(['A']).ngroup()     # group number
df.groupby(['A']).cumcount()   # running count
df.groupby('A')['B'].cumsum()
```

ADVANCED RESHAPING

```
# Pivoting / Melting
df.pivot(index='A', columns='B', values='C')
df.pivot_table(index='A', columns='B', values='C', aggfunc='sum', fill_value=0)
pd.melt(df, id_vars=['A'], value_vars=['B','C'])
pd.wide_to_long(df, stubnames='B', i='id', j='time')
# Cross-tab
pd.crosstab(df['A'], df['B'], normalize='index')
```

TIME SERIES ADVANCED

```
df.set_index('date', inplace=True)
df.asfreq('D')
df.resample('M').sum()
df.resample('W').mean()
df.shift(1)
df.diff()
df.pct_change()
df.tshift(1)   # older versions
df.rolling('7D').sum()
pd.date_range('2024-01-01', periods=10, freq='D')
pd.period_range('2024-01', periods=5, freq='M')
```

MULTIINDEX (ADVANCED)

```
df.columns = pd.MultiIndex.from_tuples([('A','x'),('A','y'),('B','z')])
df.columns.get_level_values(0)
df.stack(level=0)
df.unstack(level=1)
df.swaplevel(0,1)
df.sort_index(level=[0,1])
```

CONCAT / MERGE POWER TRICKS

```
pd.concat([df1, df2], keys=['X','Y'], names=['dataset','row'])
pd.merge_asof(df1, df2, on='time')
pd.merge_ordered(df1, df2, on='date', fill_method='ffill')
pd.concat([df1, df2], ignore_index=True)
pd.concat([df1, df2], join='inner')
```

WINDOW OPS ADVANCED

```
df['rolling_sum'] = df['A'].rolling(window=5, min_periods=2).sum()
df['exp_mean'] = df['A'].ewm(alpha=0.3).mean()
df['expand_max'] = df['A'].expanding().max()
```

SAMPLING & RANDOM

```
df.sample(frac=0.2)
df.sample(n=5, random_state=42)
df.sample(frac=0.3, replace=True)
```

ADVANCED APPLY TECHNIQUES

```
df.applymap(lambda x: str(x).upper())   # element-wise
df.transform({'A': np.sqrt, 'B': np.log})
df.aggregate({'A':['sum','mean'], 'B':['min','max']})
```

EXPANDING / CUMULATIVE OPS

```
df['cum_sum'] = df['A'].cumsum()
df['cum_prod'] = df['A'].cumprod()
df['cum_min'] = df['A'].cummin()
df['cum_max'] = df['A'].cummax()
```

CLIPPING / ROUNDING

```
df.clip(lower=0, upper=100)
df.round(2)
df['A'].round(1)
```

QUERY & EVAL

```
df.query('A > 10 & B < 50')
df.eval('C = A + B')
df.eval('A = A * 2', inplace=True)
```

CONDITIONAL REPLACEMENT

```
df['A'] = np.where(df['A']>10, 'High', 'Low')
df['B'] = df['B'].mask(df['B'] < 0, 0)
df['C'] = df['C'].where(df['C']>0, np.nan)
```

CATEGORICAL ADVANCED

```
df['grade'] = pd.Categorical(df['grade'], categories=['A','B','C','D'], ordered=True)
df['grade'].cat.codes
df['grade'].cat.categories
df['grade'].cat.add_categories(['E'])
df['grade'].cat.remove_unused_categories()
```

SPARSE DATA (Memory Optimization)

```
s = pd.Series([0,0,1,0,0], dtype='Sparse[int]')
s.sparse.density
s.sparse.to_dense()
```

IO - EXTRA FILE FORMATS

```
pd.read_feather('file.feather')
pd.read_orc('file.orc')
pd.read_sas('file.sas7bdat')
pd.read_stata('file.dta')
pd.read_sql_table('table', conn)
```

```
df.to_stata('out.dta')
df.to_feather('out.feather')
```

EXPLODE / NESTED DATA

```
df.explode('list_column')
df['json_col'].apply(pd.Series)
```

ADVANCED STYLING

```
df.style.highlight_null('red')
df.style.background_gradient(cmap='Blues')
df.style.applymap(lambda x: 'color:red' if x < 0 else 'color:black')
```

PANDAS OPTIONS & SETTINGS

```
pd.set_option('display.max_rows', 100)
pd.set_option('display.max_columns', 50)
pd.set_option('display.precision', 3)
pd.reset_option('all')
pd.get_option('display.max_rows')
```

PERFORMANCE / OPTIMIZATION

```
df.info(memory_usage='deep')
df.memory_usage(index=True, deep=True)
df.astype({'A':'int32','B':'float32'})
df.select_dtypes(exclude=['object'])
df.copy(deep=True)
```

MISC / UTILITIES

```
df.equals(df2)
df.compare(df2)
df.combine_first(df2)
df.align(df2, join='inner')
df.isin([1,2,3])
```

ADVANCED CLEANING

```
df.rename_axis('index').reset_index()
df.convert_dtypes()
df.update(df2)
df.dropna(axis=1, how='all')
df.replace({'?': np.nan})
```

PANDAS EXTENSIONS / EXPERIMENTAL

```
df.attrs['source'] = 'API'
df.attrs
pd.eval("df.A + df.B")
pd.option_context('display.max_rows', None)
```

PROFILING (EXTERNAL ADD-ON)

```
from pandas_profiling import ProfileReport
ProfileReport(df)
```

☑ **That's it — all 50 sections = COMPLETE Pandas Coverage.**
If you master all these syntaxes, you can do *anything* in data cleaning, EDA, feature engineering, or ML preprocessing.