[Pandas Data Manipulation] (Extended-CheatSheet)

1. Importing and Exporting Data

- Import pandas: `import pandas as pd`
- Read CSV: `df = pd.read_csv('file.csv')`
- Read Excel: `df = pd.read_excel('file.xlsx', sheet_name='Sheet1')`
- Read JSON: `df = pd.read_json('file.json')`
- Read SQL query: `df = pd.read_sql_query("SELECT * FROM table", connection)`
- Read HTML table: `df = pd.read_html('https://example.com/table.html')[0]`
- Read clipboard data: `df = pd.read_clipboard()`
- Write to CSV: `df.to_csv('output.csv', index=False)`
- Write to Excel: `df.to_excel('output.xlsx', index=False)`
- Write to JSON: `df.to_json('output.json', orient='records')`
- Write to SQL: `df.to_sql('table_name', connection, if_exists='replace')`
- Write to clipboard: `df.to_clipboard()`

2. Data Inspection

- Display first rows: `df.head()`
- Display last rows: `df.tail()`
- Display random sample: `df.sample(n=5)`
- Get dataframe info: `df.info()`
- Get dataframe statistics: `df.describe()`
- Get column names: `df.columns`
- Get data types: `df.dtypes`
- Get dimensions: `df.shape`
- Check for null values: `df.isnull().sum()`
- Get unique values: `df['column'].unique()`
- Get value counts: `df['column'].value_counts()`
- Get correlation matrix: `df.corr()`
- Get covariance matrix: `df.cov()`

3. Data Selection

- Select single column: `df['column']`
- Select multiple columns: `df[['column1', 'column2']]`
- Select rows by index: `df.loc[0:5]`
- Select rows by condition: `df[df['column'] > 5]`
- Select rows and columns: `df.loc[0:5, ['column1', 'column2']]`
- Select by integer position: `df.iloc[0:5, 0:2]`
- Boolean indexing: `df[df['column'].isin(['value1', 'value2'])]`
- Query method: `df.query('column > 5 and column2 == "value"')`

By: Waleed Mousa

- Select random rows: `df.sample(n=10)`
- Select every nth row: `df.iloc[::n]`

4. Data Cleaning

- Drop null values: `df.dropna()`
- Fill null values: `df.fillna(value)`
- Fill null with method: `df.fillna(method='ffill')`
- Replace values: `df.replace(old_value, new_value)`
- Remove duplicates: `df.drop_duplicates()`
- Reset index: `df.reset_index(drop=True)`
- Rename columns: `df.rename(columns={'old_name': 'new_name'})`
- Set column as index: `df.set_index('column')`
- Convert data types: `df['column'] = df['column'].astype('int64')`
- Handle outliers: `df = df[(df['column'] < df['column'].quantile(0.95)) & (df['column'] > df['column'].quantile(0.05))]`

5. Data Transformation

- Apply function to column: `df['new_column'] = df['column'].apply(lambda x: x*2)`
- Apply function to multiple columns: `df[['col1', 'col2']] = df[['col1', 'col2']].apply(lambda x: x*2)`
- Map values: `df['column'] = df['column'].map({'old1': 'new1', 'old2': 'new2'})`
- Binning: `df['binned'] = pd.cut(df['column'], bins=[0, 25, 50, 75, 100], labels=['Low', 'Medium', 'High', 'Very High'])`
- One-hot encoding: `pd.get_dummies(df, columns=['categorical_column'])`
- Normalize data: `(df - df.min()) / (df.max() - df.min())`
- Standardize data: `(df - df.mean()) / df.std()`
- Log transformation: `np.log(df)`
- Exponential transformation: `np.exp(df)`
- Square root transformation: `np.sqrt(df)`

6. String Operations

- Lowercase: `df['column'].str.lower()`
- Uppercase: `df['column'].str.upper()`
- Titlecase: `df['column'].str.title()`
- Strip whitespace: `df['column'].str.strip()`
- Replace substring: `df['column'].str.replace('old', 'new')`
- Split string: `df['column'].str.split(',')`
- Get string length: `df['column'].str.len()`

- Extract substring: `df['column'].str[0:5]`
- Pad string: `df['column'].str.pad(10, fillchar='0')`
- Check if contains: `df['column'].str.contains('substring')`

7. DateTime Operations

- Convert to datetime: `pd.to_datetime(df['date_column'])`
- Extract year: `df['date_column'].dt.year`
- Extract month: `df['date_column'].dt.month`
- Extract day: `df['date_column'].dt.day`
- Extract weekday: `df['date_column'].dt.dayofweek`
- Extract week of year: `df['date_column'].dt.isocalendar().week`
- Extract quarter: `df['date_column'].dt.quarter`
- Add time delta: `df['date_column'] + pd.Timedelta(days=1)`
- Calculate time difference: `(df['end_date'] - df['start_date']).dt.days`
- Resample time series: `df.resample('M', on='date_column').mean()`

8. Grouping and Aggregation

- Group by single column: `df.groupby('column').mean()`
- Group by multiple columns: `df.groupby(['column1', 'column2']).sum()`
- Group by with multiple aggregations: `df.groupby('column').agg({'column1': 'mean', 'column2': 'sum'})`
- Group by with custom function: `df.groupby('column').apply(lambda x: x['column2'].max() - x['column2'].min())`
- Group by with size: `df.groupby('column').size()`
- Group by with filter: `df.groupby('column').filter(lambda x: len(x) > 2)`
- Group by with transform: `df.groupby('column')['value'].transform('mean')`
- Pivot table: `pd.pivot_table(df, values='value', index='row', columns='column', aggfunc='mean')`
- Cross-tabulation: `pd.crosstab(df['column1'], df['column2'])`
- Rolling window calculations: `df['rolling_mean'] = df['column'].rolling(window=3).mean()`

9. Merging and Combining Data

- Merge on column: `pd.merge(df1, df2, on='key')`
- Merge on index: `pd.merge(df1, df2, left_index=True, right_index=True)`
- Left join: `pd.merge(df1, df2, how='left', on='key')`
- Right join: `pd.merge(df1, df2, how='right', on='key')`
- Outer join: `pd.merge(df1, df2, how='outer', on='key')`
- Concatenate vertically: `pd.concat([df1, df2], axis=0)`
- Concatenate horizontally: `pd.concat([df1, df2], axis=1)`

- Combine first: `df1.combine_first(df2)`
- Update values: `df1.update(df2)`
- Merge asof (nearest match join): `pd.merge_asof(df1, df2, on='date')`

10. Reshaping Data

- Melt dataframe: `pd.melt(df, id_vars=['id'], value_vars=['column1', 'column2'])`
- Pivot: `df.pivot(index='date', columns='category', values='value')`
- Stack: `df.stack()`
- Unstack: `df.unstack()`
- Wide to long: `pd.wide_to_long(df, stubnames='column', i=['id'], j='variable')`

11. Advanced Indexing

- Multi-level indexing: `df.set_index(['column1', 'column2'])`
- Select from multi-index: `df.loc[('level1', 'level2'), :]`
- Cross-section of multi-index: `df.xs('level1', level='column1')`
- Swap levels: `df.swaplevel('column1', 'column2')`
- Sort index: `df.sort_index()`

12. Window Functions

- Rolling mean: `df['rolling_mean'] = df['column'].rolling(window=3).mean()`
- Expanding mean: `df['expanding_mean'] = df['column'].expanding().mean()`
- Shift values: `df['previous'] = df['column'].shift(1)`
- Lag difference: `df['diff'] = df['column'].diff()`
- Percent change: `df['pct_change'] = df['column'].pct_change()`

13. Time Series Operations

- Resample by frequency: `df.resample('M', on='date').mean()`
- Offset dates: `df.index + pd.offsets.MonthEnd(1)`
- Date range: `pd.date_range(start='2023-01-01', end='2023-12-31', freq='D')`
- Business days: `pd.bdate_range(start='2023-01-01', end='2023-12-31')`
- Time zone conversion: `df['date'].dt.tz_localize('UTC').dt.tz_convert('US/Eastern')`

14. Categorical Data

- Create categorical: `df['column'] = pd.Categorical(df['column'])`
- Get categorical codes: `df['column'].cat.codes`

- Reorder categories: `df['column'].cat.reorder_categories(['cat1', 'cat2', 'cat3'], ordered=True)`
- Add new category: `df['column'].cat.add_categories('new_category')`
- Remove unused categories: `df['column'].cat.remove_unused_categories()`

15. Handling Missing Data

- Check for missing values: `df.isnull().sum()`
- Drop rows with any missing values: `df.dropna()`
- Drop columns with any missing values: `df.dropna(axis=1)`
- Fill missing with mean: `df.fillna(df.mean())`
- Fill missing with median: `df.fillna(df.median())`
- Fill missing with mode: `df.fillna(df.mode().iloc[0])`
- Fill missing with forward fill: `df.fillna(method='ffill')`
- Fill missing with backward fill: `df.fillna(method='bfill')`
- Interpolate missing values: `df.interpolate()`
- Replace inf values: `df.replace([np.inf, -np.inf], np.nan)`

16. Data Validation and Cleaning

- Remove duplicates: `df.drop_duplicates(subset=['column'], keep='first')`
- Check data types: `df.dtypes`
- Convert data types: `df['column'] = df['column'].astype('int64')`
- Handle mixed types: `pd.to_numeric(df['column'], errors='coerce')`
- Trim whitespace: `df['column'] = df['column'].str.strip()`
- Remove non-numeric characters: `df['column'] = df['column'].str.replace('[^0-9]', '')`
- Replace values: `df['column'].replace({'old': 'new'})`
- Clip values to range: `df['column'] = df['column'].clip(lower=0, upper=100)`
- Round values: `df['column'] = df['column'].round(2)`
- Normalize column names: `df.columns = df.columns.str.lower().str.replace(' ', '_')`

17. Advanced Selection and Filtering

- Select by data type: `df.select_dtypes(include=['int64', 'float64'])`
- Filter with complex conditions: `df[(df['column1'] > 5) & (df['column2'].isin(['A', 'B']))]`
- Mask values: `df['column'].mask(df['column'] < 0, 0)`
- Where condition: `df['column'].where(df['column'] > 0, 0)`
- Filter with regex: `df[df['column'].str.contains(r'^pattern$', regex=True)]`

18. Performance Optimization

- Use inplace operations: `df.dropna(inplace=True)`
- Optimize data types: `df = df.astype({'column1': 'int32', 'column2': 'category'})`
- Use chunking for large files: `pd.read_csv('large_file.csv', chunksize=10000)`
- Use generators: `(chunk for chunk in pd.read_csv('large_file.csv', chunksize=10000))`
- Use SQL query with chunksize: `pd.read_sql_query("SELECT * FROM large_table", engine, chunksize=10000)`

19. Statistical Operations

- Describe numeric columns: `df.describe()`
- Describe categorical columns: `df.describe(include=['object', 'category'])`
- Calculate correlation: `df.corr()`
- Calculate covariance: `df.cov()`
- Calculate skewness: `df.skew()`
- Calculate kurtosis: `df.kurtosis()`
- Calculate percentiles: `df.quantile([0.25, 0.5, 0.75])`
- Calculate cumulative sum: `df['cumsum'] = df['column'].cumsum()`
- Calculate cumulative max: `df['cummax'] = df['column'].cummax()`
- Calculate cumulative min: `df['cummin'] = df['column'].cummin()`

20. Advanced Grouping and Aggregation

- Group by with multiple aggregations: `df.groupby('category').agg({'column1': ['mean', 'median'], 'column2': ['min', 'max']})`
- Group by with custom aggregation: `df.groupby('category').agg({'column': lambda x: x.max() - x.min()})`
- Group by with windowing: `df.groupby('category')['value'].rolling(window=3).mean()`
- Group by with expanding window: `df.groupby('category')['value'].expanding().sum()`
- Group by with unstack: `df.groupby(['category1', 'category2'])['value'].mean().unstack()`

21. Advanced Time Series

- Resample with custom aggregation: `df.resample('M', on='date').agg({'column1': 'mean', 'column2': 'sum'})`
- Rolling window with custom function: `df['column'].rolling(window=3).apply(lambda x: x.max() - x.min())`
- Expanding window with custom function: `df['column'].expanding().apply(lambda x: x.max() - x.min())`

- Calculate year-over-year growth: `df.groupby(df.index.year)['column'].pct_change()`
- Shift business days: `df.shift(periods=1, freq='B')`
- Rolling correlation: `df['column1'].rolling(window=30).corr(df['column2'])`
- Seasonal decompose: `from statsmodels.tsa.seasonal import seasonal_decompose; result = seasonal_decompose(df['column'], model='additive')`

22. Text and String Operations

- Extract regex pattern: `df['column'].str.extract(r'(\d+)')`
- Find all regex matches: `df['column'].str.findall(r'\d+')`
- Replace regex pattern: `df['column'].str.replace(r'\d+', 'number', regex=True)`
- Split string and expand: `df['column'].str.split(',', expand=True)`
- Concatenate strings: `df['full_name'] = df['first_name'] + ' ' + df['last_name']`
- Pad strings: `df['column'].str.pad(10, side='left', fillchar='0')`
- Remove accents: `df['column'].str.normalize('NFKD').str.encode('ASCII', errors='ignore').str.decode('ASCII')`
- Count occurrences: `df['column'].str.count('pattern')`

23. Advanced Merging and Joining

- Merge with indicator: `pd.merge(df1, df2, on='key', how='outer', indicator=True)`
- Merge on multiple keys: `pd.merge(df1, df2, on=['key1', 'key2'])`
- Merge with suffixes: `pd.merge(df1, df2, on='key', suffixes=('_left', '_right'))`
- Merge with validation: `pd.merge(df1, df2, on='key', validate='one_to_one')`
- Merge closest match: `pd.merge_asof(df1, df2, on='date', direction='nearest')`
- Merge with tolerance: `pd.merge_asof(df1, df2, on='date', tolerance=pd.Timedelta('1d'))`

24. Advanced Reshaping

- Pivot with multiple values: `df.pivot(index='date', columns='category', values=['value1', 'value2'])`
- Pivot with aggregation: `df.pivot_table(values='value', index='row', columns='column', aggfunc='sum', fill_value=0)`
- Melt with id vars: `pd.melt(df, id_vars=['id', 'date'], var_name='variable', value_name='value')`
- Crosstab with margins: `pd.crosstab(df['row'], df['column'], margins=True)`

- Explode list column: `df.explode('list_column')`

25. Advanced Indexing and Selection

- Boolean indexing with isin: `df[df['column'].isin(['value1', 'value2'])]`
- Select with query method: `df.query('column1 > 5 and column2 == "value"')`
- Select with eval method: `df[df.eval('column1 > column2 + 5')]`
- Indexing with loc and boolean array: `df.loc[df['column'] > 5, 'other_column']`
- Conditional selection with numpy where: `df[np.where(df['column'] > 5, 'High', 'Low')]`

26. Memory Optimization

- Reduce memory usage: `df.select_dtypes(include=['int']).astype('int32')`
- Use categorical data type: `df['column'] = df['column'].astype('category')`
- Use sparse data structures: `df['sparse_column'] = pd.arrays.SparseArray(df['column'])`
- Use datetime64[ns] for timestamps: `df['date'] = pd.to_datetime(df['date'])`

27. Advanced Aggregation

- Aggregate with named aggregation: `df.groupby('category').agg(total=('value', 'sum'), average=('value', 'mean'))`
- Weighted average: `df.groupby('category').apply(lambda x: np.average(x['value'], weights=x['weight']))`
- First and last aggregation: `df.groupby('category').agg({'value': ['first', 'last']})`
- Aggregate with Q1 and Q3: `df.groupby('category').agg({'value': [lambda x: x.quantile(0.25), lambda x: x.quantile(0.75)]})`
- Aggregate with custom function: `df.groupby('category').agg({'value': lambda x: x.nlargest(3).mean()})`

28. Advanced Time Series Analysis

- Autocorrelation: `df['column'].autocorr(lag=1)`
- Partial autocorrelation: `from statsmodels.tsa.stattools import pacf; pacf(df['column'], nlags=10)`
- Granger causality test: `from statsmodels.tsa.stattools import grangercausalitytests; grangercausalitytests(df[['column1', 'column2']], maxlag=5)`
- ARIMA model: `from statsmodels.tsa.arima.model import ARIMA; model = ARIMA(df['column'], order=(1,1,1)).fit()`

- Prophet forecasting: `from fbprophet import Prophet; m = Prophet().fit(df[['ds', 'y']])`

29. Advanced Visualization with Pandas

- Basic line plot: `df.plot(x='date', y='value')`
- Multiple line plot: `df.plot(x='date', y=['value1', 'value2'])`
- Scatter plot: `df.plot.scatter(x='column1', y='column2')`
- Bar plot: `df.plot.bar(x='category', y='value')`
- Histogram: `df['column'].plot.hist(bins=20)`
- Box plot: `df.boxplot(column=['value1', 'value2'])`
- Heatmap: `df.pivot('row', 'column', 'value').plot.heatmap()`
- Pair plot: `pd.plotting.scatter_matrix(df)`
- Parallel coordinates: `pd.plotting.parallel_coordinates(df, 'category')`
- Andrews curves: `pd.plotting.andrews_curves(df, 'category')`