**Pandas Pro Level Cheat Sheet**

**Import Pandas**

import pandas as pd

**Data Structures:**

*Series:* **pd.Series(data, index=index)**

*DataFrame:* **pd.DataFrame(data, columns=columns)**

**Index:** An immutable array that is used to reference data.

*Example:* index = pd.Index(['one', 'two', 'three', 'four', 'five'])

**MultiIndex:** Hierarchical indexing, allowing for more than one index level on an axis.

*Example:* arrays = [np.array(['A', 'A', 'B', 'B']), np.array([1, 2, 1, 2])] multi\_index = pd.MultiIndex.from\_arrays(arrays, names=('letters', 'numbers'))

**DatetimeIndex:** A specialized Index for datetime data.

*Example:* datetime\_index = pd.date\_range('20220101', periods=6)

**PeriodIndex:** A specialized Index for Period data.

*Example:* period\_index = pd.period\_range('2022-01', periods=6, freq='M')

**TimedeltaIndex:** A specialized Index for timedelta data.

*Example:* timedelta\_index = pd.timedelta\_range('1 days', periods=5)

**Categorical:** A data type for categorical data.

*Example:* categories = pd.Categorical(['a', 'b', 'c', 'a'], categories=['a', 'b', 'c'])

**SparseDataFrame:** Efficiently represent sparse data (data with a large number of zero or missing values).

*Example:* df\_sparse = pd.SparseDataFrame({'A': [0, 0, 0, 1, 0, 2, 0]}, default\_fill\_value=0)

**Panel:** (Considered Deprecated) A three-dimensional labeled data structure. Use MultiIndex or Panel4D.

*Example (Not Recommended):* panel\_data = {'Item1': df1, 'Item2': df2, 'Item3': df3} panel = pd.Panel(panel\_data)

**Reading and Writing Data:**

*Read CSV:* **df = pd.read\_csv('file.csv')**

*Write CSV:* **df.to\_csv('output.csv', index=False)**

*Read Excel:* **df = pd.read\_excel('file.xlsx', sheet\_name='Sheet1')**

*Write Excel:* **df.to\_excel('output.xlsx', sheet\_name='Sheet1', index=False)**

**Basic DataFrame Operations:**

***Display DataFrame:*** *Print the DataFrame to view its contents.*

*Example:* ***print(df)***

***Head and Tail:*** *Display the first and last n rows of the DataFrame.*

*Example:* ***df.head(n)******df.tail(n)***

***Info and Describe:*** *Get information about the DataFrame, including data types and non-null values. Describe provides basic statistics.*

*Example:* ***df.info()******df.describe()***

***Selection and Indexing:***

***Select Column:*** *Retrieve a specific column from the DataFrame.*

*Example:* ***df['column\_name']***

***Select Row by Index:*** *Retrieve a specific row by its index.*

*Example:* ***df.loc[index]***

***Select Rows and Columns by Index:*** *Retrieve specific rows and columns based on index values.*

*Example:* ***df.loc[start\_index:end\_index, ['col1', 'col2']]***

***Filtering Data:*** *Filter rows based on a condition.*

*Example:* ***df[df['column\_name'] > value]***

***Manipulating DataFrames:***

***Add Column:*** *Add a new column to the DataFrame.*

*Example:* ***df['new\_column'] = values***

***Drop Column:*** *Remove a column from the DataFrame.*

*Example:* ***df.drop('column\_name', axis=1, inplace=True)***

***Rename Columns:*** *Rename columns in the DataFrame.*

*Example:* ***df.rename(columns={'old\_name': 'new\_name'}, inplace=True)***

**Selection and Indexing:**

***1. Selecting Columns:***

* ***By Label:******df['column\_name']***
* ***By List of Labels:******df[['col1', 'col2', 'col3']]***

***2. Selecting Rows:***

* ***By Label (loc):******df.loc[index]***
* ***By Labels Range (loc):******df.loc[start\_index:end\_index]***
* ***By Position (iloc):******df.iloc[row\_position]***

***3. Selecting Rows and Columns:***

* ***By Label (loc):******df.loc[start\_index:end\_index, ['col1', 'col2']]***
* ***By Position (iloc):******df.iloc[row\_start:row\_end, col\_start:col\_end]***

***4. Conditional Selection:***

* ***Filtering Rows Based on a Condition:******df[df['column\_name'] > value]***
* ***Filtering Rows with Multiple Conditions:******df[(df['col1'] > value1) & (df['col2'] < value2)]***

***5. Selecting by Data Type:***

* ***Selecting Numeric Columns:******df.select\_dtypes(include='number')***
* ***Selecting Categorical Columns:******df.select\_dtypes(include='category')***

***6. Indexing Techniques:***

* ***Setting a Column as Index:******df.set\_index('column\_name', inplace=True)***
* ***Resetting Index:******df.reset\_index(inplace=True)***

***7. Hierarchical Indexing (MultiIndex):***

* ***Creating a MultiIndex:******df.set\_index(['index\_col1', 'index\_col2'], inplace=True)***
* ***Selecting from a MultiIndex:******df.loc['label1', 'label2']***

***8. Slicing and Dicing:***

* ***Slicing Rows:******df[start\_index:end\_index]***
* ***Slicing Columns:******df.loc[:, 'col\_start':'col\_end']***

***9. Working with Dates and Times:***

* ***Selecting Rows Based on Date Range:******df.loc['start\_date':'end\_date']***
* ***Resampling Time Series Data:******df.resample('D').mean()***

***10. Other Useful Methods:***

* ***isin Method for Filtering:******df[df['column'].isin(['value1', 'value2'])]***
* ***query Method for Expressive Filtering:******df.query('col1 > value1 and col2 < value2')***
* ***Conditional Assignment:******df.loc[df['column'] > value, 'new\_column'] = 'New Value'***

**Manipulating DataFrames:**

***1. Add Column:***

*Add a new column to the DataFrame. Example:* ***df['new\_column'] = values***

***2. Drop Column:***

*Remove a column from the DataFrame. Example:* ***df.drop('column\_name', axis=1, inplace=True)***

***3. Rename Columns:***

*Rename columns in the DataFrame. Example:* ***df.rename(columns={'old\_name': 'new\_name'}, inplace=True)***

***4. Handling Missing Data:***

* ***Check for Missing Values:******df.isnull()******df.notnull()***
* ***Drop Missing Values:******df.dropna()***
* ***Fill Missing Values:******df.fillna(value)***

***5. Grouping and Aggregation:***

* ***Group By:******df.groupby('column\_name')***
* ***Aggregation Functions:******df['column\_name'].agg(['mean', 'sum', 'count'])***

***6. Merging and Concatenation:***

* ***Concatenate DataFrames:******pd.concat([df1, df2], axis=0, ignore\_index=True)***
* ***Merge DataFrames:******pd.merge(df1, df2, on='common\_column', how='inner')***

***7. Time Series Operations:***

* ***Convert to DateTime:******df['datetime\_column'] = pd.to\_datetime(df['datetime\_column'])***
* ***Resample Time Series:******df.resample('D').mean()***
* ***Shift and Lag:******df['lagged\_column'] = df['column'].shift(periods=n)***

***8. Pivot Tables:***

* ***Create Pivot Table:******pivot\_table = pd.pivot\_table(df, values='values', index='index\_column', columns='column\_to\_pivot', aggfunc=np.sum)***

***9. Advanced Functions:***

* ***Apply Function:******df.apply(lambda x: custom\_function(x), axis=1)***

***10. Window Functions (rolling):***

***df['rolling\_mean'] = df['column'].rolling(window=n).mean()***

***Custom Functions with np.vectorize:***

*vectorized\_function = np.vectorize(custom\_function)*

*df['new\_column'] = vectorized\_function(df['column'])*

**More on Handling Missing Data:**

***1. Checking for Missing Values:***

* ***Check for Missing Values:***
  + ***df.isnull()****: Returns a DataFrame of the same shape as* ***df*** *with* ***True*** *where missing values are present,* ***False*** *otherwise.*
  + ***df.notnull()****: Returns the opposite of* ***df.isnull()****.*

***2. Removing Missing Values:***

* ***Drop Missing Values:***
  + ***df.dropna()****: Removes rows containing any missing values.*
  + ***df.dropna(subset=['col1', 'col2'])****: Removes rows where specific columns have missing values.*
  + ***df.dropna(axis=1)****: Removes columns containing any missing values.*

***3. Filling Missing Values:***

* ***Fill Missing Values:***
  + ***df.fillna(value)****: Fills missing values with a specific constant value.*
  + ***df.fillna(df.mean())****: Fills missing values with the mean of each column.*
  + ***df.fillna(method='ffill')****: Forward fills missing values using the previous value.*
  + ***df.fillna(method='bfill')****: Backward fills missing values using the next value.*

***4. Interpolation:***

* ***Linear Interpolation:***
  + ***df.interpolate()****: Performs linear interpolation to fill missing values.*

***5. Imputation:***

* ***Simple Imputation:***
  + *Using statistical measures like mean, median, or mode to fill missing values.*
* ***Impute with Sklearn:***
  + *Using Scikit-Learn's imputation methods for more advanced imputation strategies.*

***6. Handling Missing Values in Time Series Data:***

* ***Forward and Backward Filling:***
  + *Particularly useful for time series data.*
  + ***df.ffill()****: Forward fills missing values.*
  + ***df.bfill()****: Backward fills missing values.*

***7. Handling Missing Values in Categorical Data:***

* ***Fill Categorical Missing Values:***
  + ***df['categorical\_column'].fillna(value)****: Fills missing values in a specific categorical column.*
* ***Fill Categorical with Mode:***
  + ***df['categorical\_column'].fillna(df['categorical\_column'].mode()[0])****: Fills missing values with the mode.*

***8. Handling Missing Values in Text Data:***

* ***Fill Text Data:***
  + ***df['text\_column'].fillna(value)****: Fills missing values in a text column.*

***9. Handling Missing Values in Specific Columns:***

* ***Conditional Filling:***
  + *Filling missing values in a column based on conditions.*

***10. Handling Missing Values in Grouped Data:***

* ***Grouped Imputation:***
  + *Imputing missing values based on groups in the data.*

***11. Missing Values Heatmap:***

* ***Visualizing Missing Data:***
  + *Using a heatmap to visualize the distribution of missing values.*

**More on Grouping and Aggregation:**

***1. Grouping Data:***

* *Group By Single Column:* ***grouped = df.groupby('column\_name')***
* *Group By Multiple Columns:* ***grouped = df.groupby(['col1', 'col2'])***

***2. Aggregating Data:***

* *Basic Aggregation Functions:* ***grouped['numeric\_column'].agg(['mean', 'sum', 'count'])***
* *Applying Different Aggregations to Different Columns:* ***grouped.agg({'col1': 'mean', 'col2': 'sum'})***

***3. Multiple Aggregation Functions:***

* *Using Multiple Aggregation Functions:* ***grouped['numeric\_column'].agg(['mean', 'sum', 'count'])***

***4. Custom Aggregation Functions:***

* *Defining and Using Custom Aggregation Functions:* ***grouped['numeric\_column'].agg(custom\_function)***

***5. Grouping by Time Periods:***

* *Grouping Time Series Data:* ***df.groupby(pd.Grouper(freq='M')).agg({'col1': 'mean', 'col2': 'sum'})***

***6. Grouping and Aggregating with Multiple Operations:***

* *Using* ***.agg*** *with Multiple Operations:* ***grouped['numeric\_column'].agg(['mean', 'sum', lambda x: x.max() - x.min()])***

***7. Transforming Data:***

* *Transforming Data Within Groups:* ***grouped['numeric\_column'].transform(lambda x: (x - x.mean()) / x.std())***

***8. Filtering Groups:***

* *Filtering Groups Based on Aggregated Values:* ***grouped.filter(lambda x: x['numeric\_column'].sum() > threshold)***

***9. Grouping by Index:***

* *Grouping by Index Levels:* ***df.set\_index(['col1', 'col2']).groupby(level=['col1', 'col2'])***

***10. Pivot Tables:***

* *Creating Pivot Tables:* ***pivot\_table = pd.pivot\_table(df, values='values', index='index\_column', columns='column\_to\_pivot', aggfunc=np.sum)***

***11. Named Aggregations (Pandas 0.25.0 and later):***

* *Using Named Aggregations:* ***grouped.agg(total=('numeric\_column', 'sum'), average=('numeric\_column', 'mean'))***

***12. Combining GroupBy with Other Pandas Operations:***

* *GroupBy with* ***apply****:* ***df.groupby('column\_name').apply(lambda x: custom\_function(x))***
* *GroupBy with* ***merge*** *and* ***concat****:* ***pd.merge(df, grouped, on='common\_column', how='inner')***

***13. Hierarchical Indexing (MultiIndex) with Grouping:***

* *Grouping with MultiIndex:* ***df.groupby(['col1', 'col2']).agg({'numeric\_column': 'mean'})***

**Mor on Merging and Concatenation:**

***1. Concatenation:***

* *Concatenating DataFrames vertically (along rows):*
  + ***pd.concat([df1, df2])***
  + ***pd.concat([df1, df2], ignore\_index=True)***
* *Concatenating DataFrames horizontally (along columns):*
  + ***pd.concat([df1, df2], axis=1)***

***2. Merging:***

* *Merging DataFrames based on a common column:*
  + ***pd.merge(df1, df2, on='common\_column', how='inner')***
* *Merging on multiple columns:*
  + ***pd.merge(df1, df2, on=['col1', 'col2'], how='inner')***
* *Different types of joins (****how*** *parameter):*
  + *Inner Join:* ***pd.merge(df1, df2, on='common\_column', how='inner')***
  + *Left Join:* ***pd.merge(df1, df2, on='common\_column', how='left')***
  + *Right Join:* ***pd.merge(df1, df2, on='common\_column', how='right')***
  + *Outer Join:* ***pd.merge(df1, df2, on='common\_column', how='outer')***
* *Merging on index:*
  + ***pd.merge(df1, df2, left\_index=True, right\_index=True, how='inner')***

***3. Joining:***

* *Joining DataFrames on index:*
  + ***df1.join(df2, how='inner')***
* *Joining on columns:*
  + ***df1.join(df2, on='common\_column', how='inner')***

***4. Concatenating Series:***

* *Concatenating Series along rows:*
  + ***pd.concat([series1, series2])***
* *Concatenating Series along columns:*
  + ***pd.concat([series1, series2], axis=1)***

***5. Concatenating and Merging with MultiIndex:***

* *Concatenating along MultiIndex:*
  + ***pd.concat([df1, df2], keys=['key1', 'key2'])***
* *Merging with MultiIndex:*
  + ***pd.merge(df1, df2, left\_on=['col1', 'col2'], right\_index=True, how='inner')***

**More on Time Series Operations:**

***1. Convert to DateTime:***

* *Convert a column to a datetime format:*
  + ***df['datetime\_column'] = pd.to\_datetime(df['datetime\_column'])***

***2. Resample Time Series:***

* *Resample time series data:*
  + ***df.resample('D').mean()****: Resample to daily frequency, calculating the mean.*

***3. Shifting and Lagging:***

* *Shift data in a column:*
  + ***df['lagged\_column'] = df['column'].shift(periods=n)***
* *Lagging data in a time series:*
  + *df['lagged\_column'] = df['column'].shift(periods=n)*

***4. Rolling Windows:***

* *Calculate rolling mean:*
  + ***df['rolling\_mean'] = df['column'].rolling(window=n).mean()***
* *Calculate rolling sum:*
  + ***df['rolling\_sum'] = df['column'].rolling(window=n).sum()***
* *Other rolling window functions:* ***min()****,* ***max()****,* ***std()****, etc.*

***5. Time Series Indexing:***

* *Set DateTime as index:*
  + ***df.set\_index('datetime\_column', inplace=True)***
* *Reset index:*
  + ***df.reset\_index(inplace=True)***

***6. Time Delta:***

* *Create a time delta column:*
  + ***df['time\_delta'] = df['end\_date'] - df['start\_date']***

***7. Date Range:***

* *Generate a date range:*
  + ***date\_range = pd.date\_range(start='start\_date', end='end\_date', freq='D')***

***8. Time Series Visualization:***

* *Plot time series data:*
  + ***df.plot(x='datetime\_column', y='value\_column', kind='line')***

***9. Time Zone Handling:***

* *Set time zone:*
  + ***df['datetime\_column'] = df['datetime\_column'].dt.tz\_localize('UTC').dt.tz\_convert('America/New\_York')***
* *Remove time zone information:*
  + ***df['datetime\_column'] = df['datetime\_column'].dt.tz\_localize(None)***

**Pivot Tables:**

***1. Create Pivot Table:***

* *Create a basic pivot table:*
  + ***pivot\_table = pd.pivot\_table(df, values='values', index='index\_column', columns='column\_to\_pivot', aggfunc=np.sum)***

***2. Aggregate Values:***

* *Aggregate values using different aggregation functions:*
  + ***pivot\_table = pd.pivot\_table(df, values='values', index='index\_column', columns='column\_to\_pivot', aggfunc={'values': 'sum'})***

***3. Handling Missing Values in Pivot Tables:***

* *Handling missing values during aggregation:*
  + ***pivot\_table = pd.pivot\_table(df, values='values', index='index\_column', columns='column\_to\_pivot', aggfunc=np.sum, fill\_value=0)***

***4. Multi-level Pivot Tables:***

* *Create a multi-level pivot table:*
  + ***pivot\_table = pd.pivot\_table(df, values='values', index=['index\_column1', 'index\_column2'], columns='column\_to\_pivot', aggfunc=np.sum)***

***5. Pivot Tables with Multiple Aggregation Functions:***

* *Use multiple aggregation functions:*
  + ***pivot\_table = pd.pivot\_table(df, values='values', index='index\_column', columns='column\_to\_pivot', aggfunc={'values': ['sum', 'mean']})***

***6. Pivot Tables with Custom Aggregation Functions:***

* *Use custom aggregation functions:*
  + ***pivot\_table = pd.pivot\_table(df, values='values', index='index\_column', columns='column\_to\_pivot', aggfunc={'values': custom\_function})***

***7. Grand Totals and Margins:***

* *Include grand totals and margins:*
  + ***pivot\_table = pd.pivot\_table(df, values='values', index='index\_column', columns='column\_to\_pivot', aggfunc=np.sum, margins=True, margins\_name='Total')***

***8. Pivot Tables with Date-Time Index:***

* *Create a pivot table with a date-time index:*
  + ***pivot\_table = pd.pivot\_table(df, values='values', index=pd.Grouper(freq='M', key='datetime\_column'), columns='column\_to\_pivot', aggfunc=np.sum)***

***9. Pivot Tables with Categories:***

* *Create a pivot table with categorical columns:*
  + ***pivot\_table = pd.pivot\_table(df, values='values', index='index\_column', columns='category\_column', aggfunc=np.sum)***

***10. Cross Tabulation:***

* *Create a cross-tabulation table:*
  + ***cross\_tab = pd.crosstab(index=df['index\_column'], columns=df['column\_to\_pivot'], values=df['values'], aggfunc=np.sum)***

**More on Advanced Functions:**

***1. Apply Function:***

* *Apply a function to each element or row/column of a DataFrame:*
  + ***df['new\_column'] = df['column'].apply(lambda x: custom\_function(x))***
  + ***df.apply(lambda x: custom\_function(x), axis=1)***

***2. Vectorized Operations with NumPy:***

* *Use vectorized operations for improved performance:*
  + ***df['new\_column'] = np.sqrt(df['column'])***
  + ***df['new\_column'] = np.log(df['column'])***

***3. Custom Functions with np.vectorize:***

* *Vectorize custom functions for element-wise operations:*
  + *vectorized\_function = np.vectorize(custom\_function)*
  + *df['new\_column'] = vectorized\_function(df['column'])*

***4. Window Functions (Rolling):***

* *Apply rolling window functions for time-series data:*
  + ***df['rolling\_mean'] = df['column'].rolling(window=n).mean()***
  + ***df['rolling\_sum'] = df['column'].rolling(window=n).sum()***

***5. Lambda Functions:***

* *Use lambda functions for quick, inline operations:*
  + ***df['new\_column'] = df['column'].apply(lambda x: x \* 2)***

***6. map Function:***

* *Map values based on a dictionary or function:*
  + ***df['category\_column'] = df['numeric\_column'].map({1: 'A', 2: 'B', 3: 'C'})***
  + ***df['new\_column'] = df['column'].map(lambda x: custom\_mapping\_function(x))***

***7. applymap Function:***

* *Apply a function to every element of a DataFrame:*
  + ***df.applymap(lambda x: custom\_function(x))***

***8. transform Function:***

* *Apply a function element-wise or along a specific axis:*
  + ***df['new\_column'] = df['numeric\_column'].transform(lambda x: x - x.mean())***

***9. pivot Function:***

* *Pivot the DataFrame based on the specified columns:*
  + ***df\_pivot = df.pivot(index='index\_column', columns='column\_to\_pivot', values='values')***

***10. pd.get\_dummies for One-Hot Encoding:***

* *Convert categorical variables into dummy/indicator variables:*
  + ***df\_encoded = pd.get\_dummies(df, columns=['categorical\_column'])***

***11. groupby with Custom Aggregation Functions:***

* *Use custom aggregation functions with* ***groupby****:*
  + ***grouped = df.groupby('group\_column').agg(custom\_function)***