# # [ Dataframe Operations With Pandas ] ( CheatSheet )

## 1. Creating and Reading Data

- **Creating an Empty DataFrame**: `pd.DataFrame()`
- **Creating a DataFrame from a Dictionary**: `pd.DataFrame({'col1': [1, 2], 'col2': [3, 4]})`
- **Reading CSV File**: `pd.read_csv('file.csv')`
- **Reading Excel File**: `pd.read_excel('file.xlsx')`
- **Reading JSON File**: `pd.read_json('file.json')`
- **Reading SQL Query**: `pd.read_sql_query('SELECT * FROM table', connection)`

## 2. Viewing and Inspecting Data

- **Viewing the First Few Rows**: `df.head()`
- **Viewing the Last Few Rows**: `df.tail()`
- **Getting DataFrame Info**: `df.info()`
- **Getting Summary Statistics**: `df.describe()`
- **Displaying Column Names**: `df.columns`
- **Displaying Row Indices**: `df.index`

## 3. Data Selection and Indexing

- **Selecting a Single Column**: `df['column']`
- **Selecting Multiple Columns**: `df[['col1', 'col2']]`
- **Selecting Rows by Position**: `df.iloc[10:20]`
- **Selecting Rows by Index Label**: `df.loc['index1':'index2']`
- **Conditional Selection**: `df[df['column'] > 0]`
- **Setting Index**: `df.set_index('column')`
- **Resetting Index**: `df.reset_index()`

## 4. Data Cleaning

- **Dropping Columns**: `df.drop(columns=['col'])`
- **Dropping Rows**: `df.drop(index=['index'])`
- **Renaming Columns**: `df.rename(columns={'old': 'new'})`
- **Checking for Missing Values**: `df.isnull()`
- **Dropping Missing Values**: `df.dropna()`
- **Filling Missing Values**: `df.fillna(value)`

By: Waleed Mousa

- **Converting Data Types**: `df.astype({'col': 'int'})`

## 5. Data Manipulation

- **Applying Functions**: `df.apply(lambda x: x + 1)`
- **Mapping Values**: `df['column'].map({'a': 1, 'b': 2})`
- **Replacing Values**: `df.replace({'old': 'new'})`
- **Adding New Columns**: `df['new_col'] = df['col1'] + df['col2']`
- **Deleting Columns**: `del df['column']`
- **Concatenating DataFrames**: `pd.concat([df1, df2])`
- **Merging DataFrames**: `pd.merge(df1, df2, on='col')`

## 6. Sorting and Ranking

- **Sorting by an Index**: `df.sort_index()`
- **Sorting by a Column**: `df.sort_values(by='col')`
- **Ranking Data**: `df['col'].rank()`

## 7. Unique Values, Value Counts, and Membership

- **Getting Unique Values**: `df['col'].unique()`
- **Counting Unique Values**: `df['col'].nunique()`
- **Getting Value Counts**: `df['col'].value_counts()`
- **Checking Membership**: `df['col'].isin(['val1', 'val2'])`

## 8. Grouping and Aggregation

- **Grouping Data**: `df.groupby('col')`
- **Aggregate Functions**: `df.agg({'col1': 'sum', 'col2': 'mean'})`
- **Grouping and Aggregating**: `df.groupby('col').agg({'col1': 'sum', 'col2': 'mean'})`
- **Pivot Tables**: `df.pivot_table(values='D', index='A', columns='C')`

## 9. Time Series Data

- **Convert String to DateTime**: `pd.to_datetime(df['col'])`
- **Resampling Time Series Data**: `df.resample('M').mean()`
- **Shifting Dates and Times**: `df.shift(1)`
- **Window Functions**: `df.rolling(window=5).mean()`

## 10. Visualization

- **Plotting Data**: `df.plot()`
- **Histograms**: `df['col'].hist()`
- **Box Plots**: `df.boxplot(column=['col1', 'col2'])`
- **Scatter Plots**: `df.plot.scatter(x='col1', y='col2')`

## 11. File Writing and Output

- **Writing to CSV**: `df.to_csv('file.csv')`
- **Writing to Excel**: `df.to_excel('file.xlsx')`
- **Writing to JSON**: `df.to_json('file.json')`
- **Writing to SQL Database**: `df.to_sql('table', connection)`

## 12. Advanced DataFrame Operations

- **MultiIndex / Hierarchical Index**: `df.set_index(['col1', 'col2'])`
- **Crosstab**: `pd.crosstab(df['col1'], df['col2'])`
- **Normalizing Data**: `(df - df.mean()) / df.std()`
- **Binning Data**: `pd.cut(df['col'], bins)`

## 13. Missing Data and Interpolation

- **Interpolating Missing Values**: `df.interpolate()`
- **Dropping Duplicate Rows**: `df.drop_duplicates()`
- **Replacing Outliers**: `df['col'][df['col'] > threshold] = new_value`

## 14. Combining and Reshaping Data

- **Stacking and Unstacking**: `df.stack(), df.unstack()`
- **Melting Data**: `pd.melt(df)`
- **Pivoting**: `df.pivot('row', 'col', 'values')`
- **Concatenating Along an Axis**: `pd.concat([df1, df2], axis=1)`

## 15. Advanced String Operations

- **String Methods**: `df['col'].str.upper()`
- **Regular Expressions**: `df['col'].str.extract('(pattern)', expand=True)`
- **String Replacement**: `df['col'].str.replace('old', 'new')`

## 16. Handling Large Data

- **Chunking Large Files**: `pd.read_csv('file.csv', chunksize=1000)`
- **Dask for Parallel Computing**: `import dask.dataframe as dd; dd.from_pandas(df, npartitions=10)`
- **Sampling Data**: `df.sample(frac=0.1)`

## 17. Efficiency and Performance

- **Query Method for Filtering**: `df.query('col > 0')`
- **Evaluating Expressions**: `df.eval('new_col = col1 + col2')`
- **Using Categorical Data**: `df['col'] = df['col'].astype('category')`

## 18. Memory Management

- **Reducing Memory Usage**: `df.astype('float32')`
- **Memory Usage of DataFrame**: `df.memory_usage()`

## 19. Multi-Threading and Parallel Processing

- **Parallel Apply with Dask**: `import dask.dataframe as dd; dd.from_pandas(df, npartitions=10).map_partitions(lambda df: df.apply(func))`

## 20. Dataframe Styling and Formatting

- **Styling DataFrames**: `df.style.apply(highlight_func)`
- **Setting Display Format**: `pd.options.display.float_format = '{:.2f}'.format`

## 21. Advanced Indexing and Slicing

- **Index Slicing with loc and iloc**: `df.loc['row1':'row2', 'col1':'col2']`
- **Conditional Slicing**: `df.loc[df['col'] > 0]`
- **Indexing with isin**: `df[df['col'].isin([val1, val2])]`

## 22. Data Integrity and Validation

- **Verifying Integrity**: `df.validate_subset(['col1', 'col2'])`
- **Ensuring No NA Values**: `df.dropna(subset=['col1', 'col2'])`

## 23. Advanced Merging and Joining

- **Inner Join**: `df1.merge(df2, on='col', how='inner')`
- **Outer Join**: `df1.merge(df2, on='col', how='outer')`
- **Left Join**: `df1.merge(df2, on='col', how='left')`
- **Right Join**: `df1.merge(df2, on='col', how='right')`

## 24. Data Type Conversion and Management

- **Converting Types**: `df['col'].astype('int')`
- **Handling Time Series Data Type**: `pd.to_datetime(df['date_col'])`
- **Converting to Category for Efficiency**: `df['col'].astype('category')`

## 25. Saving and Serializing Dataframes

- **Saving DataFrame to Pickle**: `df.to_pickle('df.pkl')`
- **Loading DataFrame from Pickle**: `pd.read_pickle('df.pkl')`
- **Saving to HDF5**: `df.to_hdf('data.h5', 'df')`
- **Loading from HDF5**: `pd.read_hdf('data.h5', 'df')`

## 26. Working with External Databases

- **Querying from SQL Database**: `pd.read_sql('SELECT * FROM table', connection)`
- **Writing to SQL Database**: `df.to_sql('table', connection, if_exists='replace')`

## 27. Advanced Dataframe Features

- **Using applymap for Elementwise Function**: `df.applymap(func)`
- **Expanding Data with explode**: `df.explode('list_col')`
- **Aggregating with Named Agg**: `df.groupby('col').agg(min_col=('col', 'min'), max_col=('col', 'max'))`
- **Transforming Data with transform**: `df.groupby('col').transform(lambda x: x - x.mean())`

## 28. Multi-Level Indexing (Hierarchical Indexing)

- **Creating MultiIndex from Tuples**: `pd.MultiIndex.from_tuples([('a', 1), ('a', 2)], names=['letter', 'number'])`
- **Setting MultiIndex in DataFrame**: `df.set_index(['Col1', 'Col2'])`
- **Sorting by MultiIndex**: `df.sort_index(level=0)`

- **Index Slicing with MultiIndex**: `df.loc[('index1', 'subindex1')]`
- **Stacking and Unstacking with MultiIndex**: `df.stack(), df.unstack()`

## 29. Advanced Grouping and Aggregation

- **Custom Aggregation Functions**: `df.groupby('col').agg(custom_agg_function)`
- **Named Aggregation**: `df.groupby('col').agg(total=('col2', 'sum'), average=('col2', 'mean'))`
- **Grouping with Different Functions per Column**: `df.groupby('col').agg({'col1': 'sum', 'col2': 'mean'})`
- **Transform Function with Groupby**: `df.groupby('col').transform('mean')`
- **Filtering After GroupBy**: `df.groupby('col').filter(lambda x: x['col2'].mean() > value)`

## 30. Time Series and Date Handling

- **Resampling Time Series Data**: `df.resample('M').mean()`
- **Shifting and Lagging Time Series Data**: `df.shift(1)`
- **Rolling Window Functions on Time Series**: `df.rolling(window=3).mean()`
- **Expanding Window Functions**: `df.expanding(2).sum()`
- **Custom Resampling of Time Series**: `df.resample('3T').apply(custom_resampler)`

## 31. Advanced Text and String Manipulation

- **Vectorized String Operations**: `df['col'].str.upper()`
- **Extracting Substrings**: `df['col'].str.extract(r'(regex)')`
- **Replacing Text with Regular Expression**: `df['col'].str.replace(r'[abc]', 'X')`
- **Splitting and Expanding Strings**: `df['col'].str.split('_').str[0]`
- **Aggregating Strings**: `df.groupby('col')['text'].agg(' '.join)`

## 32. Handling Missing and Duplicated Data

- **Filling Missing Values with Interpolation**: `df.interpolate()`
- **Filling Missing Values with Backward or Forward Fill**: `df.bfill(), df.ffill()`
- **Dropping Duplicates**: `df.drop_duplicates()`
- **Identifying Duplicate Data**: `df.duplicated()`
- **Counting Missing Values**: `df.isnull().sum()`

## 33. Pivot and Cross Tabulation

- **Pivot Without Aggregation**: `df.pivot(index='date', columns='col', values='val')`
- **Pivot Table with Multiple Aggregations**: `pd.pivot_table(df, values='D', index=['A', 'B'], columns=['C'], aggfunc=[np.sum, np.mean])`
- **Crosstabulation of Two Factors**: `pd.crosstab(df['A'], df['B'])`
- **Normalizing Crosstab**: `pd.crosstab(df['A'], df['B'], normalize='index')`

## 34. Styling and Display

- **Styling DataFrame Output**: `df.style.applymap(color_negative_red)`
- **Conditional Formatting**: `df.style.apply(highlight_max, axis=0)`
- **Bar Charts in DataFrame Cells**: `df.style.bar(subset=['A', 'B'], color='#d65f5f')`
- **Setting Global Display Options**: `pd.set_option('display.max_rows', 500)`

## 35. Saving and Serializing

- **Writing Data to a SQL Database**: `df.to_sql('table', conn, index=False, if_exists='append')`
- **Reading Data from SQL Database**: `pd.read_sql('SELECT * FROM table', conn)`
- **Saving DataFrame as Markdown**: `df.to_markdown()`
- **Saving DataFrame as HTML**: `df.to_html()`

## 36. Joins and Merges

- **Merging with Different Join Types**: `pd.merge(df1, df2, on='key', how='left/right/outer/inner')`
- **Joining on Index**: `df1.join(df2)`
- **Concatenating Along a Particular Axis**: `pd.concat([df1, df2], axis=1)`
- **Adding a Prefix or Suffix to Column Names**: `df.add_prefix('X_')`, `df.add_suffix('_Y')`

## 37. Visualization with Pandas

- **Line Plot**: `df.plot()`
- **Bar Plot**: `df.plot.bar()`
- **Histogram**: `df.plot.hist()`
- **Box Plot**: `df.plot.box()`

## 38. Optimization and Performance

- **Using Categories for Efficiency**: `df['col'] = df['col'].astype('category')`
- **Querying DataFrames**: `df.query('col > 0')`
- **Evaluating Expression**: `df.eval('new_col = col1 + col2')`
- **Parallelizing apply with Dask or Modin**: `import modin.pandas as pd; df.apply(func)`

## 39. Geospatial Data

- **Working with Geospatial Data**: `import geopandas as gpd; gpd.GeoDataFrame(df)`
- **Plotting Geospatial Data**: `gdf.plot()`
- **Spatial Joins**: `gpd.sjoin(gdf1, gdf2, op='within')`

## 40. Advanced DataFrame Features

- **Using Query Method for Complex Filtering**: `df.query('col > 0 & col < 10')`
- **Using eval for Efficient Operations**: `df.eval('col = col1 + col2')`
- **MultiIndex Slicing**: `df.xs(key='value', level='level2')`

## 41. Handling Large Data

- **Chunking Large Files for Reading**: `pd.read_csv('large_file.csv', chunksize=10000)`
- **Using Dask for Large DataFrames**: `import dask.dataframe as dd; dd.from_pandas(df, npartitions=10)`
- **Efficiently Combine Many Files**: `pd.concat((pd.read_csv(f) for f in files))`

## 42. Data Cleaning at Scale

- **Cleaning with replace**: `df.replace(to_replace="old_value", value="new_value")`
- **Batch Removing Missing Data**: `df.dropna(thresh=2)`