[Data Wrangling] (CheatSheet)

Data Importing

```
import pandas as pd
```

- Read CSV file: `df = pd.read_csv('file.csv')`
- Read Excel file: `df = pd.read_excel('file.xlsx')`
- Read JSON file: `df = pd.read_json('file.json')`
- Read SQL query: `df = pd.read_sql_query('SELECT * FROM table', connection)`
- Read clipboard: `df = pd.read_clipboard()`

Data Inspection

- View first n rows: `df.head(n)`
- View last n rows: `df.tail(n)`
- View random sample of n rows: `df.sample(n)`
- Get column names: `df.columns`
- Get data types: `df.dtypes`
- Get summary statistics: `df.describe()`
- Get unique values in a column: `df['column'].unique()`
- Get number of unique values in a column: `df['column'].nunique()`
- Get value counts in a column: `df['column'].value_counts()`
- Check for missing values: `df.isnull().sum()`

Data Filtering

- Filter rows based on condition: `df[df['column'] > value]`
- Filter rows based on multiple conditions: `df[(df['column1'] > value1) & (df['column2'] < value2)]`
- Filter rows based on string pattern: `df[df['column'].str.contains('pattern')]`
- Filter rows based on list of values: `df[df['column'].isin([value1, value2])]`
- Filter rows based on date range: `df[(df['date'] >= 'start_date') & (df['date'] <= 'end_date')]`
- Filter top n rows: `df.nlargest(n, 'column')`
- Filter bottom n rows: `df.nsmallest(n, 'column')`

Data Selection

By: Waleed Mousa

- Select specific columns: `df[['column1', 'column2']]`
- Select rows by index: `df.loc[index]`
- Select rows by index range: `df.loc[start_index:end_index]`
- Select rows and columns by labels: `df.loc[row_labels, column_labels]`
- Select rows by integer index: `df.iloc[index]`
- Select rows by integer index range: `df.iloc[start_index:end_index]`
- Select rows and columns by integer index: `df.iloc[row_indexes, column_indexes]`

Data Sorting

- Sort by column: `df.sort_values('column')`
- Sort by multiple columns: `df.sort_values(['column1', 'column2'])`
- Sort in descending order: `df.sort_values('column', ascending=False)`
- Sort by index: `df.sort_index()`

Data Grouping and Aggregation

- Group by column and count: `df.groupby('column').size()`
- Group by column and compute mean: `df.groupby('column').mean()`
- Group by column and compute sum: `df.groupby('column').sum()`
- Group by column and compute multiple aggregations: `df.groupby('column').agg(['mean', 'sum', 'min', 'max'])`
- Group by multiple columns and compute aggregations: `df.groupby(['column1', 'column2']).mean()`
- Group by column and apply custom function: `df.groupby('column').apply(custom_function)`
- Pivot table: `df.pivot_table(index='column1', columns='column2', values='value_column', aggfunc='mean')`

Data Merging and Joining

- Concatenate DataFrames vertically: `pd.concat([df1, df2])`
- Concatenate DataFrames horizontally: `pd.concat([df1, df2], axis=1)`
- Merge DataFrames on a common column: `pd.merge(df1, df2, on='common_column')`
- Merge DataFrames on multiple common columns: `pd.merge(df1, df2, on=['column1', 'column2'])`
- Merge DataFrames with left join: `pd.merge(df1, df2, on='common_column', how='left')`

By: Waleed Mousa

- Merge DataFrames with right join: `pd.merge(df1, df2, on='common_column', how='right')`
- Merge DataFrames with inner join: `pd.merge(df1, df2, on='common_column', how='inner')`
- Merge DataFrames with outer join: `pd.merge(df1, df2, on='common_column', how='outer')`

Data Reshaping and Pivoting

- Transpose DataFrame: `df.T`
- Reshape DataFrame from long to wide format: `df.pivot(index='id', columns='variable', values='value')`
- Reshape DataFrame from wide to long format: `df.melt(id_vars=['id'], value_vars=['variable1', 'variable2'])`
- Stack DataFrame: `df.stack()`
- Unstack DataFrame: `df.unstack()`

Data Cleaning and Preprocessing

- Remove duplicates: `df.drop_duplicates()`
- Remove duplicates based on specific columns: `df.drop_duplicates(subset=['column1', 'column2'])`
- Remove rows with missing values: `df.dropna()`
- Remove columns with missing values: `df.dropna(axis=1)`
- Fill missing values with a specific value: `df.fillna(value)`
- Fill missing values with the mean of the column: `df['column'].fillna(df['column'].mean())`
- Fill missing values with the mode of the column: `df['column'].fillna(df['column'].mode()[0])`
- Fill missing values with the previous value: `df.fillna(method='ffill')`
- Fill missing values with the next value: `df.fillna(method='bfill')`
- Replace values in a column: `df['column'].replace(old_value, new_value)`
- Rename columns: `df.rename(columns={'old_name': 'new_name'})`
- Convert column to lowercase: `df['column'] = df['column'].str.lower()`
- Convert column to uppercase: `df['column'] = df['column'].str.upper()`
- Strip whitespace from column: `df['column'] = df['column'].str.strip()`
- Convert column to datetime: `df['column'] = pd.to_datetime(df['column'])`
- Extract year from datetime column: `df['year'] = df['date'].dt.year`
- Extract month from datetime column: `df['month'] = df['date'].dt.month`
- Extract day from datetime column: `df['day'] = df['date'].dt.day`

By: Waleed Mousa

- Convert column to numeric: `df['column'] = pd.to_numeric(df['column'])`
- Convert column to categorical: `df['column'] = df['column'].astype('category')`
- One-hot encode categorical column: `pd.get_dummies(df['column'])`
- Label encode categorical column: `from sklearn.preprocessing import LabelEncoder le = LabelEncoder() df['column'] = le.fit_transform(df['column'])`

Data Visualization

```
import matplotlib.pyplot as plt import seaborn as sns
```

- Line plot: `df.plot(x='x_column', y='y_column')`
- Bar plot: `df.plot.bar(x='x_column', y='y_column')`
- Histogram: `df['column'].plot.hist()`
- Scatter plot: `df.plot.scatter(x='x_column', y='y_column')`
- Box plot: `df.boxplot(column='column')`
- Violin plot: `sns.violinplot(x='x_column', y='y_column', data=df)`
- Heatmap: `sns.heatmap(df.corr())`
- Pairplot: `sns.pairplot(df)`
- Countplot: `sns.countplot(x='column', data=df)`
- Barplot: `sns.barplot(x='x_column', y='y_column', data=df)`

Data Exporting

- Export DataFrame to CSV: `df.to_csv('file.csv', index=False)`
- Export DataFrame to Excel: `df.to_excel('file.xlsx', index=False)`
- Export DataFrame to JSON: `df.to_json('file.json')`
- Export DataFrame to SQL table: `df.to_sql('table_name', connection, if_exists='replace')`
- Export DataFrame to clipboard: `df.to_clipboard(index=False)`

Advanced Data Manipulation

- Apply function to each element in a column: `df['column'].apply(function)`
- Apply function to each row: `df.apply(function, axis=1)`
- Aggregate data using a custom function: `df.agg(function)`
- Aggregate data using multiple functions: `df.agg(['mean', 'sum', 'min', 'max'])`
- Window functions - rolling mean: `df['column'].rolling(window=n).mean()`
- Window functions - rolling sum: `df['column'].rolling(window=n).sum()`

By: Waleed Mousa

- Window functions - expanding mean: `df['column'].expanding().mean()`
- Window functions - expanding sum: `df['column'].expanding().sum()`
- Shift values in a column: `df['column'].shift(periods=n)`
- Rank values in a column: `df['column'].rank()`
- Cumulative sum of values in a column: `df['column'].cumsum()`
- Cumulative product of values in a column: `df['column'].cumprod()`
- Difference between consecutive values in a column: `df['column'].diff()`
- Percentage change between consecutive values in a column: `df['column'].pct_change()`

Data Validation

- Check if a column contains only unique values: `df['column'].is_unique`
- Check if a column contains any missing values: `df['column'].hasnans`
- Check if a column contains only numeric values: `df['column'].dtype.kind in 'iufc'`
- Check if a column contains only string values: `df['column'].dtype.kind in 'ObUS'`
- Check if a column contains only boolean values: `df['column'].dtype == bool`
- Check if a column contains only datetime values: `df['column'].dtype == np.datetime64`
- Check if a column matches a regular expression pattern: `df['column'].str.contains(regex_pattern).all()`
- Check if values in a column are within a specific range: `(df['column'] >= min_value) & (df['column'] <= max_value)`

Data Transformation

- Apply a function to multiple columns: `df[['column1', 'column2']] = df[['column1', 'column2']].apply(function)`
- Create a new column based on a condition: `df['new_column'] = np.where(df['column'] > value, 'A', 'B')`
- Bin continuous values into discrete intervals: `pd.cut(df['column'], bins=[0, 10, 20, 30])`
- Bin continuous values into quantiles: `pd.qcut(df['column'], q=4)`
- Scale values in a column to a specific range: `from sklearn.preprocessing import MinMaxScaler scaler = MinMaxScaler() df['column'] = scaler.fit_transform(df[['column']])`

By: Waleed Mousa

- Standardize values in a column (mean=0, std=1): `from sklearn.preprocessing import StandardScaler scaler = StandardScaler() df['column'] = scaler.fit_transform(df[['column']])`
- Normalize values in a column (min=0, max=1): `df['column'] = (df['column'] - df['column'].min()) / (df['column'].max() - df['column'].min())`
- Calculate the rolling median of a column: `df['column'].rolling(window=n).median()`
- Calculate the rolling standard deviation of a column: `df['column'].rolling(window=n).std()`
- Calculate the rolling correlation between two columns: `df[['column1', 'column2']].rolling(window=n).corr()`

Time Series Data Manipulation

- Convert a column to a time series index: `df.set_index('timestamp_column', inplace=True)`
- Resample time series data by a specific frequency: `df.resample('D').mean()`
- Shift time series data by a specific offset: `df.shift(periods=n)`
- Calculate the rolling mean of a time series column: `df['column'].rolling(window='30D').mean()`
- Calculate the rolling sum of a time series column: `df['column'].rolling(window='30D').sum()`
- Calculate the rolling maximum of a time series column: `df['column'].rolling(window='30D').max()`
- Calculate the rolling minimum of a time series column: `df['column'].rolling(window='30D').min()`
- Interpolate missing values in a time series column: `df['column'].interpolate()`

Advanced Visualization

- Stacked bar plot: `df.plot.bar(x='x_column', y=['y_column1', 'y_column2'], stacked=True)`
- Grouped bar plot: `df.groupby(['x_column', 'group_column']).sum()['y_column'].unstack().plot.bar()`
- Stacked area plot: `df.plot.area(x='x_column', y=['y_column1', 'y_column2'], stacked=True)`
- Grouped box plot: `df.boxplot(column='y_column', by='group_column')`

By: Waleed Mousa

- Grouped violin plot: `sns.violinplot(x='x_column', y='y_column', hue='group_column', data=df)`
- Facet grid plot: `g = sns.FacetGrid(df, col='column1', row='column2') g.map(plt.scatter, 'x_column', 'y_column')`
- Pair grid plot: `g = sns.PairGrid(df) g.map_diag(plt.hist) g.map_offdiag(plt.scatter)`
- Joint plot: `sns.jointplot(x='x_column', y='y_column', data=df)`
- KDE plot: `sns.kdeplot(df['column'])`
- Regression plot: `sns.regplot(x='x_column', y='y_column', data=df)`

Data Aggregation and Summarization

- Calculate the mean of a column grouped by another column: `df.groupby('group_column')['value_column'].mean()`
- Calculate the sum of a column grouped by another column: `df.groupby('group_column')['value_column'].sum()`
- Calculate the minimum of a column grouped by another column: `df.groupby('group_column')['value_column'].min()`
- Calculate the maximum of a column grouped by another column: `df.groupby('group_column')['value_column'].max()`
- Calculate the median of a column grouped by another column: `df.groupby('group_column')['value_column'].median()`
- Calculate the standard deviation of a column grouped by another column: `df.groupby('group_column')['value_column'].std()`
- Calculate the variance of a column grouped by another column: `df.groupby('group_column')['value_column'].var()`
- Calculate the count of values in a column grouped by another column: `df.groupby('group_column')['value_column'].count()`
- Calculate the number of unique values in a column grouped by another column: `df.groupby('group_column')['value_column'].nunique()`
- Calculate the first value in a column grouped by another column: `df.groupby('group_column')['value_column'].first()`
- Calculate the last value in a column grouped by another column: `df.groupby('group_column')['value_column'].last()`

By: Waleed Mousa