[Data Cleaning] {CheatSheet}

1. Handling Missing Values

- **Identify Missing Values**: `df.isnull().sum()`
- **Drop Rows with Missing Values**: `df.dropna()`
- **Drop Columns with Missing Values**: `df.dropna(axis=1)`
- **Fill Missing Values with a Constant**: `df.fillna(value)`
- **Fill Missing Values with Mean/Median/Mode**: `df.fillna(df.mean())`
- **Forward Fill Missing Values**: `df.ffill()`
- **Backward Fill Missing Values**: `df.bfill()`
- **Interpolate Missing Values**: `df.interpolate()`

2. Data Type Conversions

- **Convert Data Type of a Column**: `df['col'] = df['col'].astype('type')`
- **Convert to Numeric**: `pd.to_numeric(df['col'], errors='coerce')`
- **Convert to Datetime**: `pd.to_datetime(df['col'], errors='coerce')`
- **Convert to Categorical**: `df['col'] = df['col'].astype('category')`

3. Dealing with Duplicates

- **Identify Duplicate Rows**: `df.duplicated()`
- **Drop Duplicate Rows**: `df.drop_duplicates()`
- **Drop Duplicates in a Specific Column**: `df.drop_duplicates(subset='col')`
- **Drop Duplicates Keeping the Last Occurrence**: `df.drop_duplicates(keep='last')`

4. Text Data Cleaning

- **Trim Whitespace**: `df['col'] = df['col'].str.strip()`
- **Convert to Lowercase**: `df['col'] = df['col'].str.lower()`
- **Convert to Uppercase**: `df['col'] = df['col'].str.upper()`
- **Remove Specific Characters**: `df['col'] = df['col'].str.replace('[character]', '')`

By: Waleed Mousa

- **Replace Text Based on Pattern (Regex)**: `df['col'] = df['col'].str.replace(r'[regex]', 'replacement')`
- **Split Text into Columns**: `df[['col1', 'col2']] = df['col'].str.split(',', expand=True)`

5. Categorical Data Processing

- **One-Hot Encoding**: `pd.get_dummies(df['col'])`
- **Label Encoding**: `from sklearn.preprocessing import LabelEncoder; encoder = LabelEncoder(); df['col'] = encoder.fit_transform(df['col'])`
- **Map Categories to Values**: `df['col'] = df['col'].map({'cat1': 1, 'cat2': 2})`
- **Convert Category to Ordinal**: `df['col'] = df['col'].cat.codes`

6. Normalization and Scaling

- **Min-Max Scaling**: `from sklearn.preprocessing import MinMaxScaler; scaler = MinMaxScaler(); df['col'] = scaler.fit_transform(df[['col']])`
- **Standard Scaling (Z-Score)**: `from sklearn.preprocessing import StandardScaler; scaler = StandardScaler(); df['col'] = scaler.fit_transform(df[['col']])`
- **Robust Scaling (Median, IQR)**: `from sklearn.preprocessing import RobustScaler; scaler = RobustScaler(); df['col'] = scaler.fit_transform(df[['col']])`

7. Handling Outliers

- **Remove Outliers with IQR**: `Q1 = df['col'].quantile(0.25); Q3 = df['col'].quantile(0.75); IQR = Q3 - Q1; df = df[~((df['col'] < (Q1 - 1.5 * IQR)) | (df['col'] > (Q3 + 1.5 * IQR)))]`
- **Remove Outliers with Z-Score**: `from scipy import stats; df = df[np.abs(stats.zscore(df['col'])) < 3]`
- **Capping and Flooring Outliers**: `df['col'] = df['col'].clip(lower=lower_bound, upper=upper_bound)`

By: Waleed Mousa

8. Data Transformation

- **Log Transformation**: `df['col'] = np.log(df['col'])`
- **Square Root Transformation**: `df['col'] = np.sqrt(df['col'])`
- **Power Transformation (Box-Cox, Yeo-Johnson)**: `from sklearn.preprocessing import PowerTransformer; pt = PowerTransformer(method='yeo-johnson'); df['col'] = pt.fit_transform(df[['col']])`
- **Binning Data**: `df['bin_col'] = pd.cut(df['col'], bins=[range])`

9. Time Series Data Cleaning

- **Set Datetime Index**: `df.set_index('datetime_col', inplace=True)`
- **Resample Time Series Data**: `df.resample('D').mean()`
- **Fill Missing Time Series Data**: `df.asfreq('D', method='ffill')`
- **Time-Based Filtering**: `df['year'] = df.index.year; df[df['year'] > 2000]`

10. Data Frame Operations

- **Merge Data Frames**: `pd.merge(df1, df2, on='key', how='inner')`
- **Concatenate Data Frames**: `pd.concat([df1, df2], axis=0)`
- **Join Data Frames**: `df1.join(df2, on='key')`
- **Pivot Table**: `df.pivot_table(index='row', columns='col', values='value')`

11. Column Operations

- **Aggregate Functions (sum, mean, etc.)**: `df.groupby('group_col').agg({'agg_col': ['sum', 'mean']})`
- **Rolling Window Calculations**: `df['col'].rolling(window=5).mean()`
- **Expanding Window Calculations**: `df['col'].expanding().sum()`

12. Handling Complex Data Types

- **Explode List to Rows**: `df.explode('list_col')`
- **Work with JSON Columns**: `df['json_col'].apply(lambda x: json.loads(x))`

By: Waleed Mousa

- **Parse Nested Structures**: `df['new_col'] = df['struct_col'].apply(lambda x: x['nested_field'])`

13. Dealing with Geospatial Data

- **Handling Latitude and Longitude**: `df['distance'] = df.apply(lambda x: calculate_distance(x['lat'], x['long']), axis=1)`
- **Geocoding Addresses**: `df['coordinates'] = df['address'].apply(geocode_address)`

14. Data Quality Checks

- **Check for Data Consistency**: `assert df['col1'].notnull().all()`
- **Validate Data Ranges**: `df[(df['col'] >= low_val) & (df['col'] <= high_val)]`
- **Assert Data Types**: `assert df['col'].dtype == 'expected_type'`

15. Efficient Computations

- **Use Vectorized Operations**: `df['col'] = df['col1'] + df['col2']`
- **Parallel Processing with Dask**: `import dask.dataframe as dd; ddf = dd.from_pandas(df, npartitions=10); result = ddf.compute()`

16. Working with Large Datasets

- **Sampling Data for Quick Insights**: `sampled_df = df.sample(frac=0.1)`
- **Chunking Large Files for Processing**: `for chunk in pd.read_csv('large_file.csv', chunksize=10000): process(chunk)`

17. Feature Engineering

- **Creating Polynomial Features**: `from sklearn.preprocessing import PolynomialFeatures; poly = PolynomialFeatures(degree=2); df_poly = poly.fit_transform(df[['col1', 'col2']])`
- **Encoding Cyclical Features (e.g., hour of day, day of week)**: `df['hour_sin'] = np.sin(df['hour'] * (2 * np.pi / 24))`

By: Waleed Mousa

18. Data Imputation

- **Impute Missing Values with KNN**: `from sklearn.impute import KNNImputer; imputer = KNNImputer(n_neighbors=5); df['col'] = imputer.fit_transform(df[['col']])`
- **Iterative Imputation**: `from sklearn.experimental import enable_iterative_imputer; from sklearn.impute import IterativeImputer; imputer = IterativeImputer(); df_imputed = imputer.fit_transform(df)`

19. Data Validation

- **Using Pandera for Schema Validation**: `import pandera as pa; schema = pa.DataFrameSchema({'col': pa.Column(pa.Int, nullable=False)}); schema.validate(df)`
- **Validating Range of Values**: `df['col'].between(low_value, high_value)`

20. Data Anonymization

- **Hashing Sensitive Data**: `df['hashed_col'] = df['sensitive_col'].apply(lambda x: hash_function(x))`
- **Randomized Noise Addition**: `df['col'] = df['col'] + np.random.normal(0, 1, df.shape[0])`
- **Masking Values**: `df['col'] = df['col'].apply(lambda x: x[:3] + '***')`

21. Data Integration and Alignment

- **Aligning Columns from Different DataFrames**: `df1, df2 = df1.align(df2, join='inner', axis=1)`
- **Combining Data from Multiple Sources**: `df_combined = pd.merge(df1, df2, on='common_key')`

22. String Operations and Regular Expressions

- **Extracting Substrings with Regex**: `df['extracted'] = df['text_col'].str.extract(r'(pattern)')`

By: Waleed Mousa

- **Removing Unwanted Characters**: `df['clean_text'] = df['text'].str.replace('[^\w\s]', '', regex=True)`

23. Handling Time and Date

- **Extracting Date Components**: `df['year'] = df['date_col'].dt.year`
- **Calculating Date Differences**: `df['days_diff'] = (df['date_col1'] - df['date_col2']).dt.days`
- **Date Range Generation for Time Series**: `pd.date_range(start='2020-01-01', end='2020-12-31', freq='D')`

24. Working with Indexes

- **Resetting Index**: `df.reset_index(drop=True, inplace=True)`
- **Setting a Column as Index**: `df.set_index('col', inplace=True)`
- **Reindexing with a New Index**: `df.reindex(new_index)`

25. Data Compression and Memory Management

- **Reducing Memory Usage by Changing Data Types**: `df['int_col'] = df['int_col'].astype('int32')`
- **Compressing DataFrame using Categories**: `df['cat_col'] = df['cat_col'].astype('category')`

26. Handling Large and Sparse Data

- **Working with Sparse Data Structures**: `from scipy.sparse import csr_matrix; sparse_matrix = csr_matrix(df)`
- **Efficiently Storing Large Data with HDF5**: `df.to_hdf('data.h5', key='df', mode='w')`

27. Data Randomization

- **Shuffling Rows Randomly**: `df = df.sample(frac=1).reset_index(drop=True)`
- **Generating Random Samples**: `df_sample = df.sample(n=100)`

By: Waleed Mousa

28. Feature Extraction

- **Extracting Features from Text**: `from sklearn.feature_extraction.text import CountVectorizer; vectorizer = CountVectorizer(); X = vectorizer.fit_transform(df['text_col'])`
- **Dimensionality Reduction (e.g., PCA)**: `from sklearn.decomposition import PCA; pca = PCA(n_components=2); df_reduced = pca.fit_transform(df)`

29. Combining Data

- **Appending Rows of Another DataFrame**: `df = df.append(other_df)`
- **Concatenating DataFrames Vertically or Horizontally**: `pd.concat([df1, df2], axis=0)`

30. Data Cleaning Automation

- **Using Clean Function from CleanPandas**: `from cleanpandas import clean; df = clean(df)`
- **Automated Data Cleaning with DataCleaner**: `from datacleaner import autoclean; df = autoclean(df)`

31. Handling Numerical Data

- **Rounding Numeric Columns**: `df['col'] = df['col'].round(decimals=2)`
- **Discretizing Continuous Variables**: `df['binned_col'] = pd.qcut(df['col'], q=4)`

32. Geospatial Data Processing

- **Coordinate Transformation**: `df['x'], df['y'] = zip(*df['coordinates'].apply(transform_coord))`
- **Distance Calculation Between Coordinates**: `df['distance'] = df.apply(lambda row: calc_distance(row['lat1'], row['lon1'], row['lat2'], row['lon2']), axis=1)`

33. Multilingual and Locale-Specific Operations

By: Waleed Mousa

- **Converting Currencies or Units**: `df['converted_col'] = df['amount'].apply(convert_currency)`
- **Locale-Specific Sorting**: `df.sort_values(by='name', key=lambda col: col.str.normalize('NFKD'))`

34. Advanced DataFrame Manipulations

- **Pivoting and Unpivoting Data**: `df.pivot(index='date', columns='variable', values='value')`
- **Stacking and Unstacking Data**: `df.stack(); df.unstack()`

35. Custom Cleaning Functions

- **Applying Custom Cleaning Functions**: `df['clean_col'] = df['col'].apply(custom_clean_function)`
- **Using Lambda Functions for Quick Cleaning**: `df['processed_col'] = df['col'].apply(lambda x: x.strip().lower())`