# **#\_ Exploratory Data Analysis (EDA) with Pandas [CheatSheet]**

# 1. Data Loading

- Reαd CSV File: df = pd.read\_csv('filename.csv')
- Read Excel File: df = pd.read\_excel('filename.xlsx')
- Read from SQL Database: df = pd.read\_sql(query, connection)

## 2. Basic Data Inspection

- Display Top Rows: df.head()
- Display Bottom Rows: df.tail()
- Display Data Types:df.dtypes
- Summary Statistics: df.describe()
- Display Index, Columns, and Data: df.info()

#### 3. Data Cleaning

- Check for Missing Values: df.isnull().sum()
- Fill Missing Values: df.fillna(value)
- Drop Missing Values: df.dropna()
- Rename Columns: df.rename(columns={'old\_name': 'new\_name'})
- Drop Columns: df.drop(columns=['column\_name'])

#### 4. Data Transformation

- Apply Function: df['column'].apply(lambda x: function(x))
- Group By and Aggregate: df.groupby('column').agg({'column': 'sum'})
- Pivot Tables: df.pivot\_table(index='column1', values='column2', aggfunc='mean')
- Merge DataFrames: pd.merge(df1, df2, on='column')
- Concatenate DataFrames: pd.concat([df1, df2])

# 5. Data Visualization Integration

- Histogram: df['column'].hist()
- Boxplot: df.boxplot(column=['column1', 'column2'])

- Scatter Plot: df.plot.scatter(x='col1', y='col2')
- Line Plot: df.plot.line()
- Bar Chart: df['column'].value\_counts().plot.bar()

#### 6. Statistical Analysis

- Correlation Matrix: df.corr()
- Covariance Matrix: df.cov()
- Value Counts: df['column'].value\_counts()
- Unique Values in Column: df['column'].unique()
- Number of Unique Values: df['column'].nunique()

#### 7. Indexing and Selection

- Select Column: df['column']
- Select Multiple Columns: df[['col1', 'col2']]
- Select Rows by Position: df.iloc[0:5]
- Select Rows by Label: df.loc[0:5]
- Conditional Selection: df[df['column'] > value]

#### 8. Data Formatting and Conversion

- Convert Data Types: df['column'].astype('type')
- String Operations: df['column'].str.lower()
- Datetime Conversion: pd.to\_datetime(df['column'])
- Setting Index: df.set\_index('column')

#### 9. Advanced Data Transformation

- Lambda Functions: df.apply(lambda x: x + 1)
- Pivot Longer/Wider Format: df.melt(id\_vars=['col1'])
- Stack/Unstack: df.stack(), df.unstack()
- Cross Tabulations: pd.crosstab(df['col1'], df['col2'])

## 10. Handling Time Series Data

- Set Datetime Index: df.set\_index(pd.to\_datetime(df['date']))
- Resampling Data: df.resample('M').mean()

■ Rolling Window Operations: df.rolling(window=5).mean()

#### 11. File Export

- Write to CSV: df.to\_csv('filename.csv')
- Write to Excel: df.to\_excel('filename.xlsx')
- Write to SQL Database: df.to\_sql('table\_name', connection)

#### 12. Data Exploration Techniques

- Profile Report (with pandas-profiling): from pandas\_profiling import
   ProfileReport; ProfileReport(df)
- Pairplot (with seaborn): import seaborn as sns; sns.pairplot(df)
- Heatmap for Correlation (with seaborn): sns.heatmap(df.corr(), annot=True)

#### 13. Advanced Data Queries

- Query Function: df.query('column > value')
- Filtering with isin: df[df['column'].isin([value1, value2])]

## 14. Memory Optimization

- Reducing Memory Usage: df.memory\_usage(deep=True)
- Change Data Types to Save Memory: df['column'].astype('category')

## 15. Multi-Index Operations

- Creating MultiIndex: df.set\_index(['col1', 'col2'])
- Slicing on MultiIndex: df.loc[(slice('index1\_start', 'index1\_end'), slice('index2\_start', 'index2\_end'))]

# 16. Data Merging Techniques

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

#### 17. Dealing with Duplicates

- Finding Duplicates: df.duplicated()
- Removing Duplicates: df.drop\_duplicates()

#### 18. Custom Operations with Apply

 Custom Apply Functions: df.apply(lambda row: custom\_func(row['col1'], row['col2']), axis=1)

#### 19. Handling Large Datasets

- Chunking Large Files: pd.read\_csv('large\_file.csv', chunksize=1000)
- Iterating Through Data Chunks: for chunk in pd.read\_csv('file.csv', chunksize=500): process(chunk)

#### 20. Integration with Matplotlib for Custom Plots

 Custom Plotting: import matplotlib.pyplot as plt; df.plot(); plt.show()

## 21. Specialized Data Types Handling

- Working with Categorical Data: df['column'].astype('category')
- Dealing with Sparse Data: pd.arrays.SparseArray(df['column'])

## 22. Performance Tuning

- Using Swifter for Faster Apply: import swifter;
   df['column'].swifter.apply(lambda x: func(x))
- Parallel Processing with Dask: import dask.dataframe as dd; ddf = dd.from\_pandas(df, npartitions=10)

#### 23. Visualization Enhancement

- Customize Plot Style: plt.style.use('ggplot')
- Histogram with Bins Specification: df['column'].hist(bins=20)
- Boxplot Grouped by Category: df.boxplot(column='num\_column', by='cat\_column')

#### 24. Advanced Grouping and Aggregation

- **Group by Multiple Columns**: df.groupby(['col1', 'col2']).mean()
- Aggregate with Multiple Functions: df.groupby('col').agg(['mean', 'sum'])
- Transform Function: df.groupby('col').transform(lambda x: x x.mean())

#### 25. Time Series Specific Operations

- Time-Based Grouping: df.groupby(pd.Grouper(key='date\_col', freq='M')).sum()
- Shifting Series for Lag Analysis: df['column'].shift(1)
- Resample Time Series Data: df.resample('M', on='date\_col').mean()

## 26. Text Data Specific Operations

- String Contains: df[df['column'].str.contains('substring')]
- String Split: df['column'].str.split(' ', expand=True)
- Regular Expression Extraction: df['column'].str.extract(r'(regex)')

#### 27. Data Normalization and Standardization

- Min-Max Normalization: (df['column'] df['column'].min()) /
  (df['column'].max() df['column'].min())
- T-Score Standardization: (df['column'] df['column'].mean()) /
  df['column'].std()

# 28. Working with JSON and XML

- Reading JSON: df = pd.read\_json('filename.json')
- Reading XML: df = pd.read\_xml('filename.xml')

#### 29. Advanced File Handling

- Read CSV with Specific Delimiter: df = pd.read\_csv('filename.csv', delimiter=';')
- Writing to JSON: df.to\_json('filename.json')

#### 30. Dealing with Missing Data

- Interpolate Missing Values: df['column'].interpolate()
- Forward Fill Missing Values: df['column'].ffill()
- Backward Fill Missing Values: df['column'].bfill()

### 31. Data Reshaping

- Wide to Long Format: pd.wide\_to\_long(df, ['col'], i='id\_col', j='year')
- Long to Wide Formαt: df.pivot(index='id\_col', columns='year', values='col')

## 32. Categorical Data Operations

- Convert Column to Categorical: df['column'] =
   df['column'].astype('category')
- Order Categories: df['column'].cat.set\_categories(['cat1', 'cat2'], ordered=True)

#### 33. Advanced Indexing

- Reset Index: df.reset\_index(drop=True)
- Set Multiple Indexes: df.set\_index(['col1', 'col2'])
- MultiIndex Slicing: df.xs(key='value', level='level\_name')

# **34.** Efficient Computations

- Use of eval() for Efficient Operations: df.eval('col1 + col2')
- Query Method for Filtering: df.query('col1 < col2')

#### 35. Integration with SciPy and StatsModels

- Linear Regression (with statsmodels): import statsmodels.api as sm; sm.OLS(y, X).fit()
- Kurtosis and Skewness (with SciPy): from scipy.stats import kurtosis, skew; kurtosis(df['column']), skew(df['column'])

## 36. Handling Large Data Efficiently

- Dask Integration for Large Data: import dask.dataframe as dd; ddf = dd.from\_pandas(df, npartitions=10)
- Sampling Data for Quick Insights: df.sample(n=1000)

#### 37. Advanced Data Merging

- SQL-like Joins: pd.merge(df1, df2, how='left', on='col')
- Concatenating Along a Different Axis: pd.concat([df1, df2], axis=1)

## 38. Profiling Data for Quick Insights

Using Pandas Profiling for Quick Analysis: from pandas\_profiling
 import ProfileReport; report = ProfileReport(df)

## 39. Working with External Data Sources

- Reading Data from HTML: dfs = pd.read\_html('http://example.com')
- Connecting to a SQL Database: from sqlalchemy import create\_engine; engine = create\_engine('sqlite:///db.sqlite'); df = pd.read\_sql('SELECT \* FROM table\_name', engine)

## 40. Data Quality Checks

Assert Statement for Data Validation: assert
 df.notnull().all().all(), "There are missing values in the
 dataframe"