# Exploratory Data Analysis (EDA) with Pandas

### 1. Data Loading

- **Read CSV File**: `df = pd.read_csv('filename.csv')`
- **Read Excel File**: `df = pd.read_excel('filename.xlsx')`
- **Read SQL Database**: `df = pd.read_sql(query, connection)`
- **Read JSON File**: `df = pd.read_json('filename.json')`
- **Read from a URL**: `df = pd.read_csv('http://example.com/data.csv')`

### 2. Basic Data Inspection

- **Display Top Rows**: `df.head()`
- **Display Bottom Rows**: `df.tail()`
- **Display Data Types**: `df.dtypes`
- **Summary Statistics**: `df.describe()`
- **Index, Columns, Data Information**: `df.info()`
- **Shape of Data**: `df.shape`

### 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'])`
- **Remove Whitespace**: `df['column'] = df['column'].str.strip()`

### 4. Data Transformation

- **Apply Function to Column**: `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])`
- **Replace Values**: `df.replace({'old_value': 'new_value'})`

### 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**: `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 (pandas-profiling)**: `from pandas_profiling import ProfileReport; ProfileReport(df)`
- **Pairplot (seaborn)**: `import seaborn as sns; sns.pairplot(df)`
- **Heatmap for Correlation (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

- **Categorical Data**: `df['column'].astype('category')`
- **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**: `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**: `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())`
- **Z-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 Delimiter**: `df = pd.read_csv('filename.csv', delimiter=';')`
- **Writing Compressed Files**: `df.to_csv('filename.csv.gz', compression='gzip')`

### 30. Advanced Date/Time Manipulation

- **Extract Year**: `df['year'] = df['date_column'].dt.year`
- **Extract Month**: `df['month'] = df['date_column'].dt.month`
- **Extract Day**: `df['day'] = df['date_column'].dt.day`
- **Extract Hour**: `df['hour'] = df['date_column'].dt.hour`
- **Extract Weekday**: `df['weekday'] = df['date_column'].dt.weekday`
- **Time Difference Between Dates**: `df['diff'] = df['end_date'] - df['start_date']`
- **Convert to DateTime**: `df['column'] = pd.to_datetime(df['column'])`

### 31. Handling Outliers

- **Identify Outliers Using IQR**:

```
Q1 = df['column'].quantile(0.25)
Q3 = df['column'].quantile(0.75)
IQR = Q3 - Q1
df_outliers = df[((df['column'] < (Q1 - 1.5 * IQR)) | (df['column'] > (Q3 + 1.5 * IQR)))]
```

- **Winsorizing Outliers**:

```
from scipy.stats.mstats import winsorize
df['column'] = winsorize(df['column'], limits=[0.05, 0.05])
```

### 32. Efficient Memory Usage

- **Downcasting Numeric Types**:

```
df['column'] = pd.to_numeric(df['column'], downcast='float')
df['column'] = pd.to_numeric(df['column'], downcast='integer')
```

- **Categorical Type for Strings**:

```
df['column'] = df['column'].astype('category')
```

### 33. Working with Missing Data

- **Filling Missing Data with Mean/Median/Mode**:

```
df['column'].fillna(df['column'].mean(), inplace=True)
df['column'].fillna(df['column'].median(), inplace=True)
df['column'].fillna(df['column'].mode()[0], inplace=True)
```

- **Interpolate Missing Data**:

```
df['column'].interpolate(method='linear')
```

- **Drop Rows with NaN Values in Specific Column**:

```
df.dropna(subset=['column'])
```

### 34. Advanced Merging and Joining Techniques

- **Merge DataFrames on Index**:

```
pd.merge(df1, df2, left_index=True, right_index=True)
```

- **Merging with Multiple Keys**:

```
pd.merge(df1, df2, left_on=['key1', 'key2'], right_on=['key1', 'key2'])
```

- **Join with Keys and Overlapping Columns**:

```
df1.join(df2.set_index('key'), on='key')
```

### 35. Handling Categorical Data

- **Convert Column to Categorical Type**:

```
df['column'] = df['column'].astype('category')
```

- **Get Categorical Codes**:

```
df['category_codes'] = df['column'].cat.codes
```

- **Reorder Categories**:

```
df['column'] = df['column'].cat.reorder_categories(['category1',
'category2'])
```

### 36. Window and Rolling Functions

- **Rolling Mean**:

```
df['rolling_mean'] = df['column'].rolling(window=3).mean()
```

- **Expanding Window Mean**:

```
df['expanding_mean'] = df['column'].expanding().mean()
```

- **Cumulative Sum**:

```
df['cum_sum'] = df['column'].cumsum()
```

### 37. Pivoting Data

- **Pivot DataFrame**:

```
df_pivot = df.pivot(index='col1', columns='col2', values='col3')
```

- **Melt (Reverse Pivot)**:

```
df_melt = df.melt(id_vars=['col1'], value_vars=['col2', 'col3'])
```

### 38. String Operations

- **Find Substrings**: `df['column'].str.contains('substring')`
- **Replace Substring in Column**: `df['column'].str.replace('old', 'new')`
- **Extract Specific Part of String**: `df['column'].str[0:3]`
- **Regular Expression Matching**: `df['column'].str.extract(r'(pattern)')`

### 39. Applying Custom Functions

- **Row-Wise Operations**:

```
df.apply(lambda row: row['col1'] + row['col2'], axis=1)
```

- **Element-Wise Operations**:

```
df['column'] = df['column'].map(lambda x: x*2)
```

### 40. Working with Large Datasets

- **Reading in Chunks**:

```
chunks = pd.read_csv('large_data.csv', chunksize=10000)
for chunk in chunks:
    process(chunk)
```

- **Optimizing with Dask for Parallel Processing**:

```
import dask.dataframe as dd
ddf = dd.read_csv('large_data.csv')
```

### 41. Data Aggregation with GroupBy

- **Group by Single Column**:

```
df.groupby('col1').mean()
```

- **Group by Multiple Columns**:

```
df.groupby(['col1', 'col2']).agg({'col3': 'mean', 'col4': 'sum'})
```

### 42. Reshaping Data

- **Stack/Unstack Data**:

```
df_stack = df.stack()
df_unstack = df_stack.unstack()
```

- **Transpose DataFrame**: `df.T`

### 43. Integration with NumPy

- **Vectorized Operations**: `df['column'] = np.where(df['column'] > 0, 1, -1)`
- **Apply Numpy Functions**: `df['log_column'] = np.log(df['column'])`

### 44. Dealing with Time Zones

- **Convert TimeZone**:

```
df['timestamp'] = df['timestamp'].dt.tz_convert('UTC')
```

- **Localize Time Zone**:

```
df['timestamp'] = df['timestamp'].dt.tz_localize('US/Eastern')
```

### 45. DataFrame Metadata

- **Check Memory Usage**: `df.memory_usage()`
- **Number of Non-NA Cells**: `df.count()`

### 46. Advanced Sorting

- **Sort by Multiple Columns**:

```
df.sort_values(by=['col1', 'col2'], ascending=[True, False])
```

- **Sort by Index**: `df.sort_index()`

### 47. Working with Mixed Data Types

- **Convert Object Data to Numeric**:

```
df['column'] = pd.to_numeric(df['column'], errors='coerce')
```

- **Handling Mixed Types with Apply**:

```
df['column'] = df['column'].apply(lambda x: int(x) if isinstance(x, str)
else x)
```

### 48. Applying Conditions

- **Creating New Column with Conditions**:

```
df['new_col'] = np.where(df['col1'] > 0, 'Positive', 'Negative')
```

- **Using loc for Conditional Assignment**:

```
df.loc[df['col1'] > 0, 'new_col'] = 'Positive'
```