# Python for Data Science - Comprehensive Reference Guide

## 1. Variables and Data Types

### Variable Assignment

```
# Basic assignment x = 5 name = "Alice" # Multiple assignment a, b, c = 1, 2, 3 # Chained assignment x = y = z = 0 # Variable naming (PEP 8) user_name = "john" # Good userName = "john" # Avoid in Python
```

### Numeric Types

```
# Integer int_val = 42 big_int = 123456789012345678901234567890 binary = 0b1010 # 10 in decimal hex_val = 0xFF # 255 in decimal # Float float_val = 3.14 scientific = 1.5e-10 infinity = float('inf') not_a_number = float('nan') # Complex complex_val = 3 + 4j real_part = complex_val.real # 3.0 imag_part = complex_val.imag # 4.0
```

### Strings

```
# String creation single = 'Hello' double = "World" triple = """Multi-line string""" # String methods text = " Python Data Science " text.lower() # " python data science " text.upper() # " PYTHON DATA SCIENCE " text.strip() # "Python Data Science" text.replace("a", "@") # " Python D@t@ Science " words = text.strip().split() # ["Python", "Data", "Science"] " ".join(words) # "Python Data Science" # String formatting name, age = "Alice", 30 f"Name: {name}, Age: {age}" # f-strings (preferred) "Name: {}, Age: {}".format(name, age) # .format() "Name: %s, Age: %d" % (name, age) # % formatting # String slicing text = "Python" text[0] # 'P' text[-1] # 'n' text[1:4] # 'yth' text[:3] # 'Pyt' text[3:] # 'hon' text[::-1] # 'nohtyP' (reverse)
```

### Boolean and Truthiness

```
# Boolean values is_true = True is_false = False # Truthiness (False values) bool(0) # False bool("") # False bool([]) # False bool({}) # False bool(None) # False # Truthiness (True values) bool(1) # True bool("text") # True bool([1, 2]) # True bool({"a": 1}) # True # Logical operators True and False # False True or False # True not True # False # Short-circuit evaluation x = 5 x > 0 and x < 10 and expensive_function() # Won't call if x <= 0
```

## 2. Data Collections

### Lists

```
# Creation lst = [1, 2, 3] empty = [] mixed = [1, "text", 3.14] nested = [[1, 2], [3, 4]] # List comprehension squares = [x**2 for x in range(5)] evens = [x for x in range(10) if x % 2 == 0] # Indexing and slicing lst[0] # 1 (first) lst[-1] # 3 (last) lst[1:3] # [2, 3] lst[::2] # [1, 3] (every 2nd) # Methods lst.append(4) # [1, 2, 3, 4] lst.extend([5, 6]) # [1, 2, 3, 4, 5, 6] lst.insert(0, 0) # [0, 1, 2, 3, 4, 5, 6] lst.remove(3) # removes first 3 popped = lst.pop() # removes and returns last lst.sort() # sorts in-place sorted(lst) # returns new sorted list lst.reverse() # reverses in-place
```

### Tuples

```
# Creation tup = (1, 2, 3) empty = () single = (1,) # Note the comma! # Tuple unpacking point = (3, 4) x, y = point # x=3, y=4 # Multiple assignment a, b = b, a # Swap values # Named tuples from collections import namedtuple Point = namedtuple('Point', ['x', 'y']) p = Point(3, 4) p.x # 3 p.y # 4
```

### Sets

```
# Creation s = {1, 2, 3} empty = set() # Not {}! # Operations s1 = {1, 2, 3} s2 = {3, 4, 5} s1 | s2 # {1, 2, 3, 4, 5} union s1 & s2 # {3} intersection s1 - s2 # {1, 2} difference s1 ^ s2 # {1, 2, 4, 5} symmetric diff # Methods s.add(4) # Add element s.remove(2) # Remove (KeyError if not found) s.discard(2) # Remove (no error if not found)
```

### Dictionaries

```
# Creation d = {"a": 1, "b": 2} empty = {} d2 = dict(a=1, b=2) # Dict comprehension squares = {x: x**2 for x in range(5)} # Accessing d["a"] # 1 d.get("c", 0) # 0 (default if not found) # Methods d.keys() # dict_keys(['a', 'b']) d.values() # dict_values([1, 2]) d.items() # dict_items([('a', 1), ('b', 2)]) d.pop("a") # removes and returns value d.update({"c": 3}) # adds/updates entries # Dictionary merging (Python 3.9+) d1 = {"a": 1} d2 = {"b": 2} merged = d1 | d2 # {"a": 1, "b": 2}
```

## 3. Control Flow

### Conditional Statements

```
# Basic if-elif-else x = 10 if x > 0: print("Positive") elif x < 0: print("Negative") else: print("Zero") # Ternary operator result = "Positive" if x > 0 else "Non-positive" # Multiple conditions if 0 < x < 100 and x % 2 == 0: print("Even number between 0 and 100")
```

### Loops

```
# For loops for i in range(5): # 0, 1, 2, 3, 4 print(i) for i in range(1, 6): # 1, 2, 3, 4, 5 print(i) for i in range(0, 10, 2): # 0, 2, 4, 6, 8 print(i) # Iterating over collections fruits = ["apple", "banana", "cherry"] for fruit in fruits: print(fruit) # Enumerate for index and value for i, fruit in enumerate(fruits): print(f"{i}: {fruit}") # Zip for parallel iteration names = ["Alice", "Bob", "Charlie"] ages = [25, 30, 35] for name, age in zip(names, ages): print(f"{name} is {age} years old") # While loops count = 0 while count < 5: print(count) count += 1 # Loop control for i in range(10): if i == 3: continue # Skip iteration if i == 7: break # Exit loop print(i) else: print("Loop completed normally") # Only if no break
```

### Comprehensions

```
# List comprehensions squares = [x**2 for x in range(10)] evens = [x for x in range(20) if x % 2 == 0] words = ["hello", "world", "python"] lengths = [len(word) for word in words] # Nested comprehensions matrix = [[i*j for j in range(3)] for i in range(3)] # [[0, 0, 0], [0, 1, 2], [0, 2, 4]] flattened = [item for row in matrix for item in row] # [0, 0, 0, 0, 1, 2, 0, 2, 4] # Dictionary comprehensions word_lengths = {word: len(word) for word in words} squared_dict = {x: x**2 for x in range(5)} # Set comprehensions unique_lengths = {len(word) for word in words} # Conditional in comprehensions positive_squares = [x**2 for x in range(-5, 6) if x > 0]
```

**Common Pitfalls:** Mutable default arguments in functions, modifying list while iterating, confusing `is` vs `==`, integer division behavior in Python 2 vs 3.

# Functions and Advanced Python

## 4. Functions

### Function Definition

```
# Basic function def greet(name): """Return a greeting message.""" return f"Hello, {name}!" result = greet("Alice") # "Hello, Alice!" # Multiple return values def get_name_age(): return "Bob", 25 name, age = get_name_age() # Tuple unpacking # Default parameters def power(base, exponent=2): return base ** exponent power(5) # 25 (5^2) power(5, 3) # 125 (5^3) # Keyword arguments def create_profile(name, age, city="Unknown"): return {"name": name, "age": age, "city": city} profile = create_profile(name="Alice", age=30, city="NYC")
```

### Advanced Parameters

```
# *args - variable positional arguments def sum_all(*args): return sum(args) sum_all(1, 2, 3, 4) # 10 # **kwargs - variable keyword arguments def create_dict(**kwargs): return kwargs create_dict(a=1, b=2, c=3) # {'a': 1, 'b': 2, 'c': 3} # Combined parameters (order matters!) def complex_func(required, default="value", *args, **kwargs): print(f"Required: {required}") print(f"Default: {default}") print(f"Args: {args}") print(f"Kwargs: {kwargs}") # Argument unpacking numbers = [1, 2, 3, 4] print(*numbers) # Same as print(1, 2, 3, 4) data = {"name": "Alice", "age": 30} create_profile(**data) # Unpacks dictionary as keyword args
```

### Lambda Functions

```
# Basic lambda square = lambda x: x**2 square(5) # 25 # Lambda with multiple arguments add = lambda x, y: x + y add(3, 4) # 7 # Lambda in higher-order functions numbers = [1, 2, 3, 4, 5] squared = list(map(lambda x: x**2, numbers)) evens = list(filter(lambda x: x % 2 == 0, numbers)) # Sorting with lambda students = [("Alice", 85), ("Bob", 90), ("Charlie", 78)] students.sort(key=lambda x: x[1]) # Sort by grade
```

### Variable Scope

```
# Global vs Local scope global_var = "I'm global" def my_function(): local_var = "I'm local" print(global_var) # Can access global print(local_var) # Can access local # Global keyword counter = 0 def increment(): global counter counter += 1 # Nonlocal keyword (for nested functions) def outer(): x = 10 def inner(): nonlocal x x += 1 inner() return x # Returns 11 # LEGB Rule: Local, Enclosing, Global, Built-in
```

## 5. File I/O and Error Handling

### File Operations

```
# Reading files with open('data.txt', 'r') as file: content = file.read() # Read entire file with open('data.txt', 'r') as file: lines = file.readlines() # Read all lines as list with open('data.txt', 'r') as file: for line in file: # Memory efficient iteration print(line.strip()) # Writing files with open('output.txt', 'w') as file: file.write("Hello, World!\n") file.writelines(["Line 1\n", "Line 2\n"]) # Appending to files with open('log.txt', 'a') as file: file.write("New log entry\n") # File modes # 'r' - read (default) # 'w' - write (overwrites) # 'a' - append # 'r+' - read and write # 'rb', 'wb' - binary modes
```

### Exception Handling

```
# Basic try-except try: result = 10 / 0 except ZeroDivisionError: print("Cannot divide by zero!") # Multiple exception types try: value = int(input("Enter a number: ")) result = 10 / value except ValueError: print("Invalid number format") except ZeroDivisionError: print("Cannot divide by zero") # Catch multiple exceptions try: # risky code pass except (ValueError, TypeError) as e: print(f"Error occurred: {e}") # Try-except-else-finally try: file = open('data.txt', 'r') except FileNotFoundError: print("File not found") else: # Executes if no exception occurred data = file.read() finally: # Always executes if 'file' in locals(): file.close() # Raising exceptions def validate_age(age): if age < 0: raise ValueError("Age cannot be negative") if age > 150: raise ValueError("Age seems unrealistic") return age
```

## 6. Built-in Functions and Standard Modules

### Essential Built-ins

```
# Type and conversion functions type(42) # isinstance(42, int) # True len("hello") # 5 str(42) # "42" int("42") # 42 float("3.14") # 3.14 bool(1) # True # Sequence functions numbers = [3, 1, 4, 1, 5] min(numbers) # 1 max(numbers) # 5 sum(numbers) # 14 sorted(numbers) # [1, 1, 3, 4, 5] reversed(numbers) # reverse iterator # Iteration functions list(enumerate(["a", "b", "c"])) # [(0, 'a'), (1, 'b'), (2, 'c')] list(zip([1, 2, 3], ["a", "b", "c"])) # [(1, 'a'), (2, 'b'), (3, 'c')] list(filter(lambda x: x > 2, numbers)) # [3, 4, 5] list(map(lambda x: x**2, [1, 2, 3])) # [1, 4, 9]
```

### Math Module

```
import math # Constants math.pi # 3.141592653589793 math.e # 2.718281828459045 # Basic functions math.sqrt(16) # 4.0 math.pow(2, 3) # 8.0 math.ceil(4.3) # 5 math.floor(4.7) # 4 math.fabs(-5) # 5.0 # Trigonometric math.sin(math.pi/2) # 1.0 math.cos(0) # 1.0 math.tan(math.pi/4) # 1.0 # Logarithmic math.log(math.e) # 1.0 math.log10(100) # 2.0 math.log2(8) # 3.0
```

### Random Module

```
import random # Random numbers random.random() # 0.0 to 1.0 random.randint(1, 6) # 1 to 6 inclusive random.uniform(1, 10) # 1.0 to 10.0 # Random choice fruits = ["apple", "banana", "cherry"] random.choice(fruits) # Random fruit random.choices(fruits, k=2) # With replacement random.sample(fruits, 2) # Without replacement # Shuffle numbers = [1, 2, 3, 4, 5] random.shuffle(numbers) # Modifies in-place # Seed for reproducibility random.seed(42)
```

### Datetime Module

```
from datetime import datetime, date, time # Current date and time now = datetime.now() today = date.today() # Creating specific dates birthday = date(1990, 5, 15) meeting = datetime(2023, 12, 25, 14, 30) # Formatting now.strftime("%Y-%m-%d %H:%M:%S") # "2023-06-15 14:30:45" # Parsing date_str = "2023-06-15" parsed = datetime.strptime(date_str, "%Y-%m-%d") # Date arithmetic from datetime import timedelta tomorrow = today + timedelta(days=1) next_week = today + timedelta(weeks=1)
```

### OS Module

```
import os # File system operations os.getcwd() # Current directory os.listdir('.') # List directory contents os.path.exists('file.txt') # Check if exists os.path.isfile('file.txt') # Check if file os.path.isdir('folder') # Check if directory # Path operations os.path.join('folder', 'subfolder', 'file.txt') os.path.dirname('/path/to/file.txt') # '/path/to' os.path.basename('/path/to/file.txt') # 'file.txt' # Environment variables home = os.environ.get('HOME') path = os.environ.get('PATH')
```

**Performance Tip:** Use list comprehensions instead of loops when possible. Use `enumerate()` instead of `range(len())`. Consider using `collections.defaultdict` for grouping operations.

# Data Science Libraries

## 7. NumPy - Numerical Computing

### Array Creation

```
import numpy as np # From lists arr = np.array([1, 2, 3, 4, 5]) matrix = np.array([[1, 2, 3], [4, 5, 6]]) # Special arrays zeros = np.zeros((3, 4)) # 3x4 array of
zeros ones = np.ones((2, 3)) # 2x3 array of ones empty = np.empty((2, 2)) # Uninitialized array identity = np.eye(3) # 3x3 identity matrix # Range arrays range_arr
= np.arange(0, 10, 2) # [0, 2, 4, 6, 8] linspace = np.linspace(0, 1, 5) # [0, 0.25, 0.5, 0.75, 1] # Random arrays random_arr = np.random.rand(3, 3) # Uniform [0,1)
normal_arr = np.random.randn(3, 3) # Standard normal randint_arr = np.random.randint(0, 10, (3, 3)) # Random integers # Array properties arr.shape # (5,) arr.size
# 5 arr.ndim # 1 arr.dtype # dtype('int64')
```

### Array Operations

```
# Indexing and slicing arr = np.array([0, 1, 2, 3, 4, 5]) arr[0] # 0 arr[-1] # 5 arr[1:4] # array([1, 2, 3]) arr[::2] # array([0, 2, 4]) # 2D indexing matrix =
np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) matrix[0, 1] # 2 matrix[1, :] # array([4, 5, 6]) matrix[:, 2] # array([3, 6, 9]) # Boolean indexing arr = np.array([1,
2, 3, 4, 5]) mask = arr > 3 arr[mask] # array([4, 5]) arr[arr % 2 == 0] # array([2, 4]) # Reshaping arr = np.arange(12) arr.reshape(3, 4) # 3x4 matrix
arr.reshape(-1, 2) # Auto-calculate rows arr.flatten() # 1D array arr.ravel() # 1D view (if possible)
```

### Mathematical Operations

```
# Element-wise operations a = np.array([1, 2, 3, 4]) b = np.array([5, 6, 7, 8]) a + b # array([6, 8, 10, 12]) a - b # array([-4, -4, -4, -4]) a * b # array([5, 12,
21, 32]) a / b # array([0.2, 0.33, 0.43, 0.5]) a ** 2 # array([1, 4, 9, 16]) # Broadcasting arr = np.array([[1, 2, 3], [4, 5, 6]]) arr + 10 # Adds 10 to all
elements arr * np.array([1, 2, 3]) # Multiplies each row # Statistical functions data = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]) np.mean(data) # 5.5
np.median(data) # 5.5 np.std(data) # 2.87 np.var(data) # 8.25 np.min(data) # 1 np.max(data) # 10 np.sum(data) # 55 np.prod(data) # 3628800 # Axis operations matrix
= np.array([[1, 2, 3], [4, 5, 6]]) np.sum(matrix, axis=0) # [5, 7, 9] (column sums) np.sum(matrix, axis=1) # [6, 15] (row sums) np.mean(matrix, axis=0) # [2.5,
3.5, 4.5] # Linear algebra A = np.array([[1, 2], [3, 4]]) B = np.array([[5, 6], [7, 8]]) np.dot(A, B) # Matrix multiplication A @ B # Alternative syntax (Python
3.5+) np.linalg.inv(A) # Matrix inverse np.linalg.det(A) # Determinant eigenvals, eigenvecs = np.linalg.eig(A) # Eigenvalues/vectors
```

## 8. Pandas - Data Analysis

### Data Structures and Creation

```
import pandas as pd # Series creation s = pd.Series([1, 2, 3, 4, 5]) s_named = pd.Series([1, 2, 3], index=['a', 'b', 'c']) # DataFrame creation df = pd.DataFrame({
'Name': ['Alice', 'Bob', 'Charlie'], 'Age': [25, 30, 35], 'City': ['NYC', 'LA', 'Chicago'] }) # From dictionary of lists data = { 'A': [1, 2, 3], 'B': [4, 5, 6],
'C': [7, 8, 9] } df = pd.DataFrame(data) # From list of dictionaries records = [ {'name': 'Alice', 'age': 25}, {'name': 'Bob', 'age': 30} ] df =
pd.DataFrame(records)
```

### Data Import/Export

```
# Reading files df = pd.read_csv('data.csv') df = pd.read_csv('data.csv', sep=';', header=0, index_col=0) df = pd.read_excel('data.xlsx', sheet_name='Sheet1') df =
pd.read_json('data.json') # Writing files df.to_csv('output.csv', index=False) df.to_excel('output.xlsx', sheet_name='Data', index=False) df.to_json('output.json',
orient='records') # CSV options df = pd.read_csv('data.csv', sep=',', # Separator header=0, # Header row index_col=0, # Index column usecols=['A', 'B'], # Specific
columns dtype={'A': 'int64'}, # Data types parse_dates=['date'], # Parse dates na_values=['NULL', 'N/A'] # Missing values )
```

### Data Inspection

```
# Basic info df.head() # First 5 rows df.tail(3) # Last 3 rows df.info() # Column info and types df.describe() # Statistical summary df.shape # (rows, columns)
df.columns # Column names df.index # Row indices df.dtypes # Data types # Missing data df.isnull() # Boolean mask of missing values df.isnull().sum() # Count of
missing values per column df.isna() # Same as isnull() # Memory usage df.memory_usage(deep=True) # Memory consumption df.select_dtypes(include=['object']).columns
# Object columns
```

### Data Selection and Filtering

```
# Column selection df['Name'] # Single column (Series) df[['Name', 'Age']] # Multiple columns (DataFrame) # Row selection df.loc[0] # Row by label df.iloc[0] # Row
by position df.loc[0:2] # Rows 0 to 2 (inclusive) df.iloc[0:3] # Rows 0 to 2 (exclusive) # Boolean indexing df[df['Age'] > 25] # Rows where Age > 25 df[df['City']
== 'NYC'] # Rows where City is NYC # Multiple conditions df[(df['Age'] > 25) & (df['City'] == 'NYC')] df[(df['Age'] < 25) | (df['Age'] > 35)] # Query method
df.query('Age > 25 and City == "NYC"') df.query('Age > @threshold') # Use variable # Advanced selection df.loc[df['Age'] > 25, ['Name', 'City']] # Specific rows
and columns df.iloc[1:3, 0:2] # Slice rows and columns by position
```

### Data Manipulation

```
# Adding/dropping columns df['Salary'] = [50000, 60000, 70000] # Add column df.drop('City', axis=1) # Drop column df.drop([0, 1], axis=0) # Drop rows # Sorting
df.sort_values('Age') # Sort by Age df.sort_values(['City', 'Age'], ascending=[True, False]) df.sort_index() # Sort by index # Grouping and aggregation grouped =
df.groupby('City') grouped.mean() # Mean by group grouped.agg(['mean', 'std', 'count']) # Multiple functions grouped['Age'].agg(['min', 'max']) # Specific column #
Apply functions df['Age'].apply(lambda x: x * 2) # Apply to Series df.apply(lambda row: row['Age'] * 2, axis=1) # Apply to rows # String operations
df['Name'].str.lower() # Lowercase df['Name'].str.contains('A') # Contains substring df['Name'].str.split() # Split strings df['Name'].str.len() # String length #
Handling missing data df.fillna(0) # Fill with 0 df.fillna(method='ffill') # Forward fill df.fillna(df.mean()) # Fill with mean df.dropna() # Drop rows with any
missing df.dropna(subset=['Age']) # Drop if Age is missing # Merging DataFrames df1 = pd.DataFrame({'A': [1, 2], 'B': [3, 4]}) df2 = pd.DataFrame({'A': [1, 2],
'C': [5, 6]}) merged = pd.merge(df1, df2, on='A') # Inner join pd.merge(df1, df2, on='A', how='left') # Left join # Concatenating pd.concat([df1, df2], axis=0) #
Stack vertically pd.concat([df1, df2], axis=1) # Stack horizontally
```

## 9. Matplotlib - Data Visualization

### Basic Plotting

```
import matplotlib.pyplot as plt # Line plot x = [1, 2, 3, 4, 5] y = [2, 4, 6, 8, 10] plt.plot(x, y) plt.show() # Multiple lines plt.plot(x, y, label='Line 1')
plt.plot(x, [1, 3, 5, 7, 9], label='Line 2') plt.legend() # Scatter plot plt.scatter(x, y) plt.scatter(x, y, s=50, c='red', alpha=0.7) # Bar plot categories =
['A', 'B', 'C', 'D'] values = [23, 45, 56, 78] plt.bar(categories, values) plt.barh(categories, values) # Horizontal # Histogram data = np.random.randn(1000)
plt.hist(data, bins=30, alpha=0.7) plt.hist(data, bins=30, density=True) # Density # Box plot data = [np.random.randn(100), np.random.randn(100)] plt.boxplot(data)
```

### Plot Customization

```
# Labels and titles plt.xlabel('X-axis Label') plt.ylabel('Y-axis Label') plt.title('Plot Title') # Axis limits plt.xlim(0, 10) plt.ylim(-5, 5) # Grid
plt.grid(True) plt.grid(True, alpha=0.3, linestyle='--') # Ticks plt.xticks([0, 2, 4, 6, 8, 10]) plt.yticks(range(-5, 6)) plt.xticks(rotation=45) # Rotate labels #
Colors and styles plt.plot(x, y, color='red') plt.plot(x, y, 'r--') # Red dashed plt.plot(x, y, 'bo-') # Blue circles with line # Line styles: '-', '--', '-.', ':'
# Markers: 'o', 's', '^', 'v', '*', '+' # Colors: 'r', 'g', 'b', 'c', 'm', 'y', 'k' # Figure size plt.figure(figsize=(10, 6)) plt.plot(x, y)
```

### Subplots and Advanced

```
# Subplots fig, axes = plt.subplots(2, 2, figsize=(10, 8)) axes[0, 0].plot(x, y) axes[0, 1].scatter(x, y) axes[1, 0].bar(categories, values) axes[1, 1].hist(data)
# Alternative subplot syntax plt.subplot(2, 2, 1) plt.plot(x, y) plt.subplot(2, 2, 2) plt.scatter(x, y) # Tight layout plt.tight_layout() # Saving plots
plt.savefig('plot.png', dpi=300, bbox_inches='tight') plt.savefig('plot.pdf', format='pdf') # Styles plt.style.use('seaborn') plt.style.use('ggplot')
available_styles = plt.style.available # Object-oriented interface fig, ax = plt.subplots() ax.plot(x, y) ax.set_xlabel('X Label') ax.set_ylabel('Y Label')
ax.set_title('Title')
```

## 10. Common Data Science Workflows

### Data Loading and Cleaning

```
# Typical workflow import pandas as pd import numpy as np import matplotlib.pyplot as plt # Load data df = pd.read_csv('data.csv') # Initial exploration
print(df.shape) print(df.info()) print(df.describe()) print(df.head()) # Check for missing values missing = df.isnull().sum() print(missing[missing > 0]) # Handle
missing values df['column'].fillna(df['column'].mean(), inplace=True) df.dropna(subset=['important_column'], inplace=True) # Data type conversions
df['date_column'] = pd.to_datetime(df['date_column']) df['category'] = df['category'].astype('category') # Remove duplicates df.drop_duplicates(inplace=True) #
Outlier detection Q1 = df['numeric_column'].quantile(0.25) Q3 = df['numeric_column'].quantile(0.75) IQR = Q3 - Q1 lower_bound = Q1 - 1.5 * IQR upper_bound = Q3 +
1.5 * IQR outliers = df[(df['numeric_column'] < lower_bound) | (df['numeric_column'] > upper_bound)]
```

### Exploratory Data Analysis

```
# Statistical summaries df.describe(include='all') df['category'].value_counts() df.corr() # Correlation matrix # Groupby analysis df.groupby('category')
['value'].agg(['mean', 'std', 'count']) # Cross-tabulation pd.crosstab(df['category1'], df['category2']) # Time series analysis (if date column exists)
df.set_index('date_column', inplace=True) monthly_avg = df.resample('M').mean() rolling_avg = df['value'].rolling(window=7).mean() # Basic visualization
df['numeric_column'].hist(bins=30) df.boxplot(column='value', by='category') df.plot.scatter(x='x_col', y='y_col', c='color_col', colormap='viridis') # Correlation
heatmap import seaborn as sns # If available correlation_matrix = df.corr() plt.figure(figsize=(10, 8)) sns.heatmap(correlation_matrix, annot=True,
cmap='coolwarm') # Distribution plots plt.figure(figsize=(12, 4)) plt.subplot(1, 3, 1) df['col1'].hist() plt.subplot(1, 3, 2) df['col2'].hist() plt.subplot(1, 3,
3) df['col3'].hist() plt.tight_layout()
```

**Memory Warning:** Large datasets can consume significant memory. Use `df.dtypes` to check data types and convert to more efficient types when possible (e.g., `int64` to `int32`, `object` to `category`).

**Performance Tips:** Use vectorized operations instead of loops. Prefer `pd.read_csv()` with `chunksize` for large files. Use `.loc` and `.iloc` for explicit indexing. Consider using `query()` for complex filtering.

## Quick Reference Tables

| Operation | NumPy | Pandas |
|---|---|---|
| Create array/series | `np.array([1,2,3])` | `pd.Series([1,2,3])` |
| Shape | `arr.shape` | `df.shape` |
| Mean | `np.mean(arr)` | `df.mean()` |
| Filter | `arr[arr > 5]` | `df[df['col'] > 5]` |

| String Method | Description | Example |
|---|---|---|
| `.lower()` | Convert to lowercase | `"HELLO".lower()` → "hello" |
| `.strip()` | Remove whitespace | `" hello ".strip()` → "hello" |
| `.split()` | Split string | `"a,b,c".split(",")` → ["a","b","c"] |
| `.replace()` | Replace substring | `"hello".replace("l","x")` → "hexxo" |