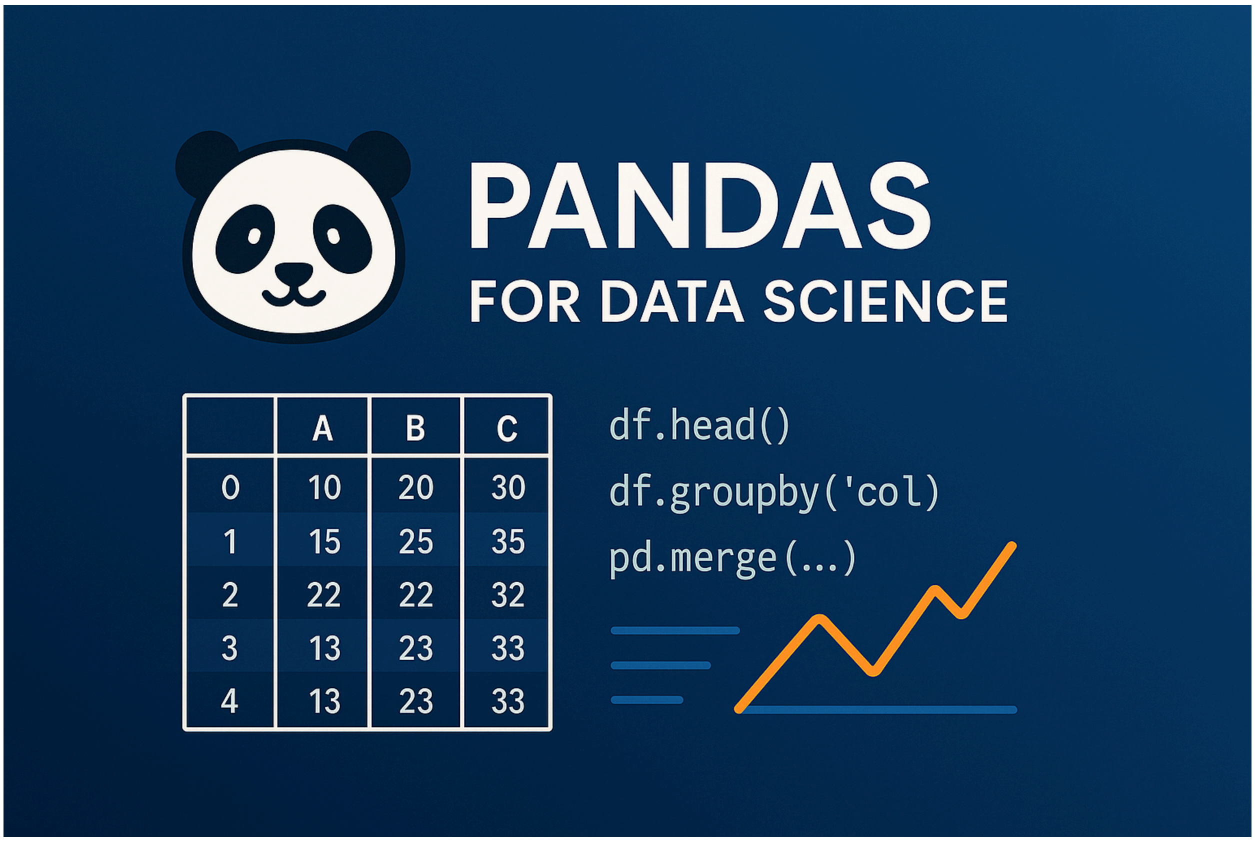
Complete Pandas for Data Science Cheat Sheet

Type	@datasciencebrain



Import Pandas

```
import pandas as pd
import numpy as np
```

Data Structures

Series (1D)

```
# Create Series
s = pd.Series([1, 2, 3, 4, 5])
s = pd.Series([1, 2, 3], index=['a', 'b', 'c'])
s = pd.Series({'a': 1, 'b': 2, 'c': 3})

# Series properties
s.values      # Underlying array
s.index       # Index labels
s.dtype       # Data type
s.shape       # Dimensions
s.size        # Number of elements
```

DataFrame (2D)

```
# Create DataFrame
df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})
df = pd.DataFrame([[1, 2], [3, 4]], columns=['A', 'B'])

# DataFrame properties
df.shape       # (rows, columns)
df.size        # Total elements
df.ndim        # Number of dimensions
df.columns     # Column names
df.index       # Row index
df.dtypes      # Data types of columns
df.info()      # Comprehensive info
df.describe()  # Statistical summary
```

Reading and Writing Data

Reading Files

```
# CSV files
df = pd.read_csv('file.csv')
```

```
df = pd.read_csv('file.csv', sep=';', header=0, index_col=0)

# Excel files
df = pd.read_excel('file.xlsx', sheet_name='Sheet1')

# JSON files
df = pd.read_json('file.json')

# SQL databases
df = pd.read_sql('SELECT * FROM table', connection)

# Other formats
df = pd.read_parquet('file.parquet')
df = pd.read_pickle('file.pkl')
df = pd.read_html('url')[0]  # Read HTML tables
```

Writing Files

```
# CSV files
df.to_csv('output.csv', index=False)
df.to_csv('output.csv', sep=';', encoding='utf-8')

# Excel files
df.to_excel('output.xlsx', sheet_name='Data', index=False)

# JSON files
df.to_json('output.json', orient='records')

# Other formats
df.to_parquet('output.parquet')
df.to_pickle('output.pkl')
df.to_sql('table_name', connection, if_exists='replace')
```

Data Inspection

Basic Inspection

```
df.head()          # First 5 rows
df.head(10)        # First 10 rows
df.tail()          # Last 5 rows
df.sample(5)       # 5 random rows
df.info()          # Data types and memory usage
df.describe()      # Statistical summary
df.nunique()       # Number of unique values per column
df.count()         # Non-null count per column
```

Data Types and Memory

```
df.dtypes          # Data types
df.memory_usage()   # Memory usage
df.select_dtypes(include=['object'])    # Select by data type
df.select_dtypes(exclude=['object'])    # Exclude by data type

# Convert data types
df['column'] = df['column'].astype('int64')
df['column'] = pd.to_numeric(df['column'], errors='coerce')
df['date'] = pd.to_datetime(df['date'])
```

Data Selection and Indexing

Column Selection

```
df['column']       # Single column (Series)
df[['col1', 'col2']] # Multiple columns (DataFrame)

# Column operations
df.column          # Dot notation (if valid Python identifier)
df.columns.tolist() # Get column names as list
```

Row Selection

```
df.iloc[0]         # First row by position
df.iloc[0:3]       # First 3 rows by position
df.iloc[-1]        # Last row

df.loc[0]          # First row by label
df.loc[0:2]        # Rows 0 to 2 by label
df.loc['index_name'] # Row by index name
```

Boolean Indexing

```
# Single condition
df[df['column'] > 5]
df[df['column'] == 'value']
df[df['column'].isin(['A', 'B', 'C'])]

# Multiple conditions
df[(df['col1'] > 5) & (df['col2'] < 10)]
df[(df['col1'] > 5) | (df['col2'] < 10)]
df[~(df['column'] == 'value')]  # NOT condition

# String operations
df[df['column'].str.contains('pattern')]
df[df['column'].str.startswith('A')]
df[df['column'].str.endswith('Z')]
```

Advanced Selection

```
# loc and iloc
df.loc[rows, columns]
df.iloc[row_positions, column_positions]

# Examples
df.loc[0:2, 'A':'C']            # Rows 0-2, columns A-C
```

```
df.iloc[0:3, 0:2]                # First 3 rows, first 2 columns
df.loc[df['A'] > 5, ['B', 'C']]   # Conditional row selection with specific columns

# Query method
df.query('A > 5 and B < 10')
df.query('column in @my_list')    # Use external variable
```

Data Cleaning

Missing Data

```
# Detect missing data
df.isnull()          # Boolean DataFrame of null values
df.isna()            # Same as isnull()
df.notnull()         # Boolean DataFrame of non-null values
df.isnull().sum()    # Count of null values per column
df.isnull().any()    # Columns with any null values

# Handle missing data
df.dropna()                    # Drop rows with any null values
df.dropna(axis=1)              # Drop columns with any null values
df.dropna(subset=['col1'])     # Drop rows with null in specific column
df.dropna(thresh=2)            # Drop rows with less than 2 non-null values

df.fillna(0)                   # Fill null values with 0
df.fillna(method='ffill')      # Forward fill
df.fillna(method='bfill')      # Backward fill
df.fillna(df.mean())           # Fill with mean
df['col'].fillna(df['col'].mode()[0])  # Fill with mode

# Interpolation
df.interpolate()               # Linear interpolation
df.interpolate(method='polynomial', order=2)  # Polynomial interpolation
```

Duplicate Data

```
# Detect duplicates
df.duplicated()              # Boolean Series of duplicate rows
df.duplicated(subset=['col']) # Check duplicates in specific column
df.duplicated().sum()        # Count of duplicate rows

# Handle duplicates
df.drop_duplicates()         # Remove duplicate rows
df.drop_duplicates(subset=['col'], keep='first')  # Keep first occurrence
df.drop_duplicates(keep='last')  # Keep last occurrence
```

String Operations

```
# Common string operations
df['col'].str.lower()        # Convert to lowercase
df['col'].str.upper()        # Convert to uppercase
df['col'].str.title()        # Title case
df['col'].str.strip()        # Remove whitespace
df['col'].str.replace('old', 'new')  # Replace strings

# String splitting
df['col'].str.split(' ')      # Split by space
df['col'].str.split(' ', expand=True)  # Split into separate columns

# String extraction
df['col'].str.extract(r'(\d+)')  # Extract digits using regex
df['col'].str.findall(r'\d+')    # Find all matches
```

Data Transformation

Adding and Modifying Columns

```
# Add new columns
df['new_col'] = df['A'] + df['B']
df['new_col'] = df['A'].apply(lambda x: x * 2)
df.assign(new_col=df['A'] * 2)

# Modify existing columns
df['A'] = df['A'] * 2
df.loc[df['A'] > 5, 'B'] = 'High'

# Conditional column creation
df['category'] = np.where(df['A'] > 5, 'High', 'Low')
df['category'] = df['A'].apply(lambda x: 'High' if x > 5 else 'Low')

# Multiple conditions
conditions = [df['A'] > 10, df['A'] > 5]
choices = ['Very High', 'High']
df['category'] = np.select(conditions, choices, default='Low')
```

Apply Functions

```
# Apply to single column
df['A'].apply(lambda x: x ** 2)
df['A'].apply(np.sqrt)

# Apply to multiple columns
df[['A', 'B']].apply(lambda x: x.max() - x.min())
df.apply(lambda row: row['A'] + row['B'], axis=1)  # Row-wise

# Map values
df['A'].map({1: 'One', 2: 'Two', 3: 'Three'})
df['A'].replace({1: 'One', 2: 'Two'})
```

Data Type Conversions

```
# Convert data types
df['col'].astype('int64')
df['col'].astype('float64')
df['col'].astype('category')
df['col'].astype(str)

# Datetime conversions
df['date'] = pd.to_datetime(df['date'])
df['date'] = pd.to_datetime(df['date'], format='%Y-%m-%d')
df['numeric'] = pd.to_numeric(df['numeric'], errors='coerce')
```

Grouping and Aggregation

GroupBy Operations

```
# Basic grouping
grouped = df.groupby('column')
grouped.mean()          # Mean of each group
grouped.sum()           # Sum of each group
grouped.count()         # Count of each group
grouped.size()          # Size of each group (including NaN)

# Multiple grouping columns
df.groupby(['col1', 'col2']).mean()

# Multiple aggregations
df.groupby('col').agg({
    'A': 'mean',
    'B': 'sum',
    'C': ['min', 'max', 'std']
})

# Custom aggregations
df.groupby('col').agg(
```

```
    mean_A=('A', 'mean'),
    sum_B=('B', 'sum'),
    custom=('C', lambda x: x.max() - x.min())
)
```

Advanced GroupBy

```
# Transform (keeps original shape)
df.groupby('col')['A'].transform('mean')  # Add group mean to each row

# Filter groups
df.groupby('col').filter(lambda x: len(x) > 5)  # Groups with >5 rows

# Apply custom functions
df.groupby('col').apply(lambda x: x.describe())

# Multiple operations
result = (df.groupby('col')
         .agg({'A': 'mean', 'B': 'sum'})
         .reset_index())
```

Pivot Tables and Cross-tabulation

Pivot Tables

```
# Basic pivot table
pd.pivot_table(df, values='A', index='col1', columns='col2')

# Multiple values and aggregations
pd.pivot_table(df,
              values=['A', 'B'],
              index='col1',
              columns='col2',
              aggfunc={'A': 'mean', 'B': 'sum'},
```

```
             fill_value=0)

# Pivot with multiple indices
pd.pivot_table(df,
             values='A',
             index=['col1', 'col2'],
             columns='col3')
```

Cross-tabulation

```
# Basic crosstab
pd.crosstab(df['col1'], df['col2'])

# With percentages
pd.crosstab(df['col1'], df['col2'], normalize=True)
pd.crosstab(df['col1'], df['col2'], normalize='index')  # Row percentages

# With values
pd.crosstab(df['col1'], df['col2'], values=df['A'], aggfunc='mean')
```

Merging and Joining

Concatenation

```
# Vertical concatenation (stack rows)
pd.concat([df1, df2])
pd.concat([df1, df2], ignore_index=True)

# Horizontal concatenation (side by side)
pd.concat([df1, df2], axis=1)

# With keys
pd.concat([df1, df2], keys=['first', 'second'])
```

Merging

```
# Inner join (default)
pd.merge(df1, df2, on='key')
pd.merge(df1, df2, on=['key1', 'key2'])

# Different join types
pd.merge(df1, df2, on='key', how='left')    # Left join
pd.merge(df1, df2, on='key', how='right')   # Right join
pd.merge(df1, df2, on='key', how='outer')   # Full outer join

# Different column names
pd.merge(df1, df2, left_on='key1', right_on='key2')

# Index-based merging
pd.merge(df1, df2, left_index=True, right_index=True)
```

Join Method

```
# Join on index
df1.join(df2)
df1.join(df2, how='outer')
df1.join(df2, rsuffix='_right')  # Handle duplicate column names
```

Time Series Operations

DateTime Operations

```
# Convert to datetime
df['date'] = pd.to_datetime(df['date'])

# Extract date components
df['year'] = df['date'].dt.year
df['month'] = df['date'].dt.month
```

```
df['day'] = df['date'].dt.day
df['weekday'] = df['date'].dt.day_name()
df['quarter'] = df['date'].dt.quarter

# Date arithmetic
df['date'] + pd.Timedelta(days=30)
df['date'] + pd.DateOffset(months=1)
```

Time Series Indexing

```
# Set datetime index
df.set_index('date', inplace=True)

# Time-based selection
df['2023']                  # All data from 2023
df['2023-01']               # January 2023
df['2023-01-01':'2023-01-31'] # Date range

# Resampling
df.resample('M').mean()     # Monthly mean
df.resample('D').sum()      # Daily sum
df.resample('H').last()     # Hourly last value
```

Statistical Operations

Descriptive Statistics

```
df.mean()          # Mean
df.median()        # Median
df.mode()          # Mode
df.std()           # Standard deviation
df.var()           # Variance
df.min()           # Minimum
df.max()           # Maximum
```

```python
df.quantile(0.25)   # 25th percentile
df.skew()           # Skewness
df.kurtosis()       # Kurtosis

# Correlation
df.corr()           # Correlation matrix
df['A'].corr(df['B']) # Correlation between two columns
df.corrwith(df['A']) # Correlation with one column
```

Window Functions

```python
# Rolling window
df['A'].rolling(window=3).mean()    # 3-period moving average
df['A'].rolling(window=5).std()     # 5-period rolling standard deviation

# Expanding window
df['A'].expanding().mean()          # Cumulative mean
df['A'].expanding().sum()           # Cumulative sum

# Exponential weighted moving average
df['A'].ewm(span=10).mean()
```

Advanced Operations

MultiIndex

```python
# Create MultiIndex
df.set_index(['col1', 'col2'])

# Access MultiIndex levels
df.index.get_level_values(0)
df.index.names

# Reset index
```

```python
df.reset_index()
df.reset_index(level=1)

# Stack and unstack
df.stack()          # Pivot columns to rows
df.unstack()        # Pivot rows to columns
```

Categorical Data

```python
# Create categorical
df['category'] = df['category'].astype('category')
df['category'] = pd.Categorical(df['category'],
                               categories=['Low', 'Medium', 'High'],
                               ordered=True)

# Categorical operations
df['category'].cat.categories
df['category'].cat.codes
df['category'].value_counts()
```

Performance Optimization

```python
# Efficient operations
df.eval('new_col = A + B')          # Faster arithmetic
df.query('A > 5')                   # Faster filtering

# Memory optimization
df.info(memory_usage='deep')        # Detailed memory usage
df.select_dtypes(include=['object']).astype('category')  # Convert to category
```

Common Data Science Patterns

Data Preprocessing Pipeline

```
def preprocess_data(df):
    # Handle missing values
    df = df.fillna(df.mean())

    # Remove duplicates
    df = df.drop_duplicates()

    # Convert data types
    for col in df.select_dtypes(include=['object']).columns:
        df[col] = df[col].astype('category')

    # Create derived features
    df['feature_ratio'] = df['feature1'] / df['feature2']

    return df
```

Feature Engineering

```
# Binning continuous variables
df['age_group'] = pd.cut(df['age'], bins=[0, 18, 35, 50, 100],
                        labels=['Child', 'Young', 'Adult', 'Senior'])

# One-hot encoding
pd.get_dummies(df['category'])
pd.get_dummies(df['category'], prefix='cat')

# Label encoding
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['category_encoded'] = le.fit_transform(df['category'])
```

Data Validation

```
# Check for data quality issues
def validate_data(df):
    print(f"Shape: {df.shape}")
    print(f"Duplicates: {df.duplicated().sum()}")
    print(f"Missing values:\n{df.isnull().sum()}")
    print(f"Data types:\n{df.dtypes}")

    # Check for outliers using IQR
    numeric_cols = df.select_dtypes(include=[np.number]).columns
    for col in numeric_cols:
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1
        outliers = df[(df[col] < Q1 - 1.5*IQR) | (df[col] > Q3 + 1.5*IQR)]
        print(f"Outliers in {col}: {len(outliers)}")
```

Quick Reference

Essential Methods Checklist

- **Data Loading**: `pd.read_csv()`, `pd.read_excel()`
- **Inspection**: `df.head()`, `df.info()`, `df.describe()`
- **Selection**: `df['col']`, `df.loc[]`, `df.iloc[]`
- **Filtering**: `df[df['col'] > value]`
- **Grouping**: `df.groupby('col').agg()`
- **Merging**: `pd.merge()`, `pd.concat()`
- **Missing Data**: `df.dropna()`, `df.fillna()`
- **Apply Functions**: `df.apply()`, `df['col'].map()`

Common Gotchas

- Use `.copy()` when modifying DataFrames to avoid SettingWithCopyWarning

- Remember that `.loc[]` is inclusive of both endpoints
- Use `errors='coerce'` with `pd.to_numeric()` for robust conversion
- Always check data types after reading files with `df.dtypes`
- Use `pd.concat()` instead of `df.append()` (deprecated)