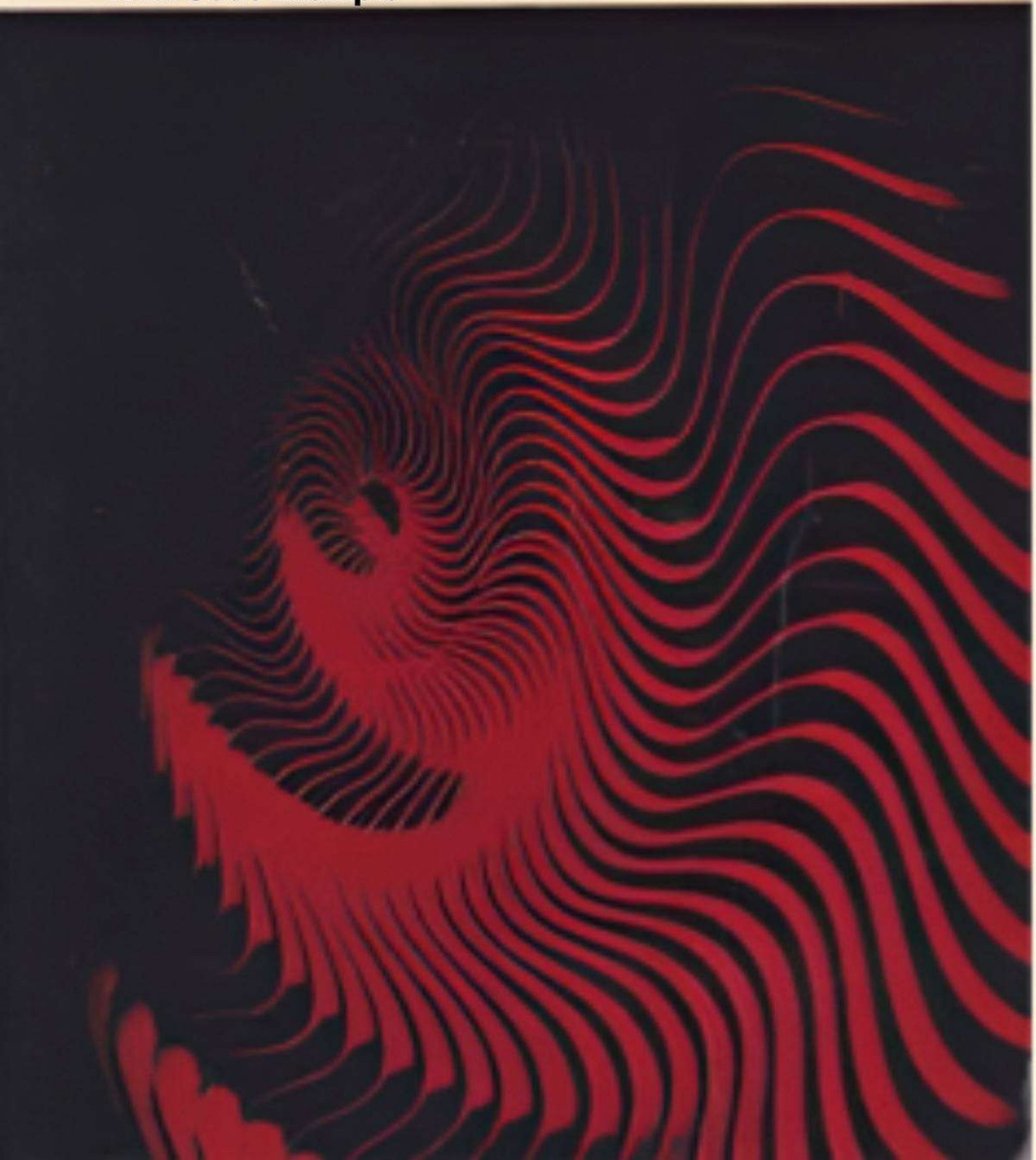
Pandas DataFrame Cheat Sheet

With Code Example



save for later

Introduction to Pandas DataFrame

A DataFrame is a 2-dimensional labeled data structure in Pandas, similar to a spreadsheet or SQL table. It's the most commonly used Pandas object, capable of holding various data types in columns.

```
import pandas as pd

# Creating a DataFrame from a dictionary
data = {'Name': ['Alice', 'Bob', 'Charlie'],
        'Age': [25, 30, 35],
        'City': ['New York', 'San Francisco', 'Los Angeles']}
df = pd.DataFrame(data)
print(df)
```

Swipe next ⟶

Creating a DataFrame

DataFrames can be created from various data sources, including dictionaries, lists, and external files.

```
# From a list of dictionaries
data = [{'a': 1, 'b': 2}, {'a': 5, 'b': 10, 'c': 20}]
df1 = pd.DataFrame(data)

# From a NumPy array
import numpy as np
arr = np.random.rand(3, 2)
df2 = pd.DataFrame(arr, columns=['A', 'B'])

print("DataFrame from list of dictionaries:\n", df1)
print("\nDataFrame from NumPy array:\n", df2)
```

Swipe next ⟶

save for later

Accessing Data in a DataFrame

DataFrame elements can be accessed using various methods, including column names, row indices, and boolean indexing.

```
df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6], 'C': [7, 8, 9]})

# Accessing a column
print("Column 'A':\n", df['A'])

# Accessing a row by index
print("\nRow at index 1:\n", df.iloc[1])

# Boolean indexing
print("\nRows where 'A' > 1:\n", df[df['A'] > 1])
```

Swipe next ⟶

follow for more

Basic DataFrame Operations

Pandas provides various operations to manipulate and analyze data in DataFrames.

```
df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6], 'C': [7, 8, 9]})

# Adding a new column
df['D'] = df['A'] + df['B']

# Applying a function to a column
df['E'] = df['A'].apply(lambda x: x ** 2)

# Basic statistics
print("DataFrame:\n", df)
print("\nMean of each column:\n", df.mean())
print("\nSum of each column:\n", df.sum())
```

Swipe next ⟶

save for later

Handling Missing Data

Pandas offers methods to detect, remove, or fill missing data in DataFrames.

```
import numpy as np

df = pd.DataFrame({'A': [1, 2, np.nan, 4],
                   'B': [5, np.nan, np.nan, 8],
                   'C': [9, 10, 11, 12]})

print("Original DataFrame:\n", df)

# Dropping rows with any NaN values
print("\nAfter dropping NaN:\n", df.dropna())

# Filling NaN values
print("\nAfter filling NaN with 0:\n", df.fillna(0))

# Interpolating missing values
print("\nAfter interpolation:\n", df.interpolate())
```

Swipe next ⟶

follow for more

Merging and Joining DataFrames

Pandas provides various methods to combine DataFrames, similar to SQL joins.

```
df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2'],
                    'B': ['B0', 'B1', 'B2']},
                    index=['K0', 'K1', 'K2'])

df2 = pd.DataFrame({'C': ['C0', 'C1', 'C2'],
                    'D': ['D0', 'D1', 'D2']},
                    index=['K0', 'K2', 'K3'])

# Concatenating DataFrames
print("Concatenated:\n", pd.concat([df1, df2], axis=1))

# Merging DataFrames
df3 = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'],
                    'A': ['A0', 'A1', 'A2', 'A3'],
                    'B': ['B0', 'B1', 'B2', 'B3']})

df4 = pd.DataFrame({'key': ['K0', 'K1', 'K2'],
                    'C': ['C0', 'C1', 'C2'],
                    'D': ['D0', 'D1', 'D2']})

print("\nMerged:\n", pd.merge(df3, df4, on='key'))
```

Swipe next ⟶

save for later

Grouping and Aggregating Data

GroupBy operations allow you to split the data, apply a function, and combine the results.

```
df = pd.DataFrame({'A': ['foo', 'bar', 'foo', 'bar', 'foo', 'bar'],
                   'B': ['one', 'one', 'two', 'three', 'two', 'two'],
                   'C': [1, 2, 3, 4, 5, 6],
                   'D': [10, 20, 30, 40, 50, 60]})

# Grouping by column 'A' and calculating mean
print("Group by 'A' and calculate mean:\n", df.groupby('A').mean())

# Grouping by multiple columns and aggregating
print("\nGroup by 'A' and 'B', then aggregate:\n",
      df.groupby(['A', 'B']).agg({'C': 'sum', 'D': 'mean'}))
```

Swipe next ⟶

save for later

Time Series Data

Pandas excels at handling time series data with its powerful datetime functionality.

```
# Creating a time series
dates = pd.date_range('20230101', periods=6)
ts = pd.Series(np.random.randn(6), index=dates)

print("Time series:\n", ts)

# Resampling time series data
print("\nMonthly mean:\n", ts.resample('M').mean())

# Rolling statistics
print("\n7-day rolling mean:\n", ts.rolling(window=7).mean())
```

Swipe next ⟶

follow for more

Data Visualization with Pandas

Pandas integrates well with Matplotlib, allowing for quick and easy data visualization.

```
import matplotlib.pyplot as plt

df = pd.DataFrame(np.random.randn(10, 4), columns=['A', 'B', 'C', 'D'])

# Line plot
df.plot(figsize=(10, 6))
plt.title('Line Plot of DataFrame Columns')
plt.show()

# Bar plot
df.iloc[5].plot(kind='bar')
plt.title('Bar Plot of Row 5')
plt.show()
```

Swipe next ⟶

save for later

Real-Life Example: Analyzing Weather Data

Let's analyze a simple weather dataset to demonstrate Pandas functionality.

```
# Sample weather data
data = {
    'date': pd.date_range('20230101', periods=10),
    'temperature': [20, 22, 19, 21, 25, 23, 22, 20, 19, 24],
    'humidity': [65, 70, 60, 68, 75, 72, 70, 65, 62, 71],
    'wind_speed': [10, 8, 12, 9, 7, 11, 10, 13, 11, 8]
}

weather_df = pd.DataFrame(data)

# Calculate average temperature and humidity
avg_temp = weather_df['temperature'].mean()
avg_humidity = weather_df['humidity'].mean()

print(f"Average Temperature: {avg_temp:.2f}°C")
print(f"Average Humidity: {avg_humidity:.2f}%")

# Find the day with highest wind speed
max_wind_day = weather_df.loc[weather_df['wind_speed'].idxmax()]
print(f"\nDay with highest wind speed: {max_wind_day['date'].strftime('%Y-%m-
%d')}")
print(f"Wind speed: {max_wind_day['wind_speed']} km/h")

# Plot temperature over time
weather_df.plot(x='date', y='temperature', figsize=(10, 6))
plt.title('Temperature Trend')
plt.ylabel('Temperature (°C)')
plt.show()
```

Swipe next ⟶

Real-Life Example: Analyzing Book Ratings

Let's analyze a dataset of book ratings to showcase Pandas' data manipulation capabilities.

```
# Sample book ratings data
data = {
    'book_id': [1, 1, 1, 2, 2, 2, 3, 3, 3],
    'user_id': [101, 102, 103, 101, 103, 104, 102, 103, 105],
    'rating': [4, 5, 3, 2, 4, 3, 5, 4, 4],
    'genre': ['Fiction', 'Fiction', 'Fiction', 'Non-Fiction', 'Non-Fiction', 'Non-
Fiction', 'Mystery', 'Mystery', 'Mystery']
}

ratings_df = pd.DataFrame(data)

# Calculate average rating per book
avg_ratings = ratings_df.groupby('book_id')['rating'].mean().reset_index()
print("Average ratings per book:\n", avg_ratings)

# Find the most popular genre
genre_popularity = ratings_df.groupby('genre').size().sort_values(ascending=False)
print("\nGenre popularity:\n", genre_popularity)

# Calculate the average rating by genre
avg_rating_by_genre = ratings_df.groupby('genre')
['rating'].mean().sort_values(ascending=False)
print("\nAverage rating by genre:\n", avg_rating_by_genre)

# Visualize average ratings by genre
avg_rating_by_genre.plot(kind='bar', figsize=(10, 6))
plt.title('Average Rating by Genre')
plt.ylabel('Average Rating')
plt.show()
```

Swipe next ⟶

Performance Optimization

save for later

Let's analyze a dataset of book ratings to showcase Pandas' data manipulation capabilities.

```
# Sample book ratings data
data = {
    'book_id': [1, 1, 1, 2, 2, 2, 3, 3, 3],
    'user_id': [101, 102, 103, 101, 103, 104, 102, 103, 105],
    'rating': [4, 5, 3, 2, 4, 3, 5, 4, 4],
    'genre': ['Fiction', 'Fiction', 'Fiction', 'Non-Fiction', 'Non-Fiction', 'Non-
Fiction', 'Mystery', 'Mystery', 'Mystery']
}

ratings_df = pd.DataFrame(data)

# Calculate average rating per book
avg_ratings = ratings_df.groupby('book_id')['rating'].mean().reset_index()
print("Average ratings per book:\n", avg_ratings)

# Find the most popular genre
genre_popularity = ratings_df.groupby('genre').size().sort_values(ascending=False)
print("\nGenre popularity:\n", genre_popularity)

# Calculate the average rating by genre
avg_rating_by_genre = ratings_df.groupby('genre')
['rating'].mean().sort_values(ascending=False)
print("\nAverage rating by genre:\n", avg_rating_by_genre)

# Visualize average ratings by genre
avg_rating_by_genre.plot(kind='bar', figsize=(10, 6))
plt.title('Average Rating by Genre')
plt.ylabel('Average Rating')
plt.show()
```

Swipe next ⟶

Performance Optimization

Pandas offers various techniques to optimize performance when working with large datasets.

```
import time

# Generate a large DataFrame
large_df = pd.DataFrame(np.random.randn(1000000, 4), columns=list('ABCD'))

# Measure time for iterating rows
start = time.time()
for index, row in large_df.iterrows():
    pass
print(f"Time for iterrows(): {time.time() - start:.2f} seconds")

# Measure time for vectorized operation
start = time.time()
large_df['A'] + large_df['B']
print(f"Time for vectorized operation: {time.time() - start:.2f} seconds")

# Using .loc for faster column access
start = time.time()
large_df.loc[:, 'A']
print(f"Time for .loc access: {time.time() - start:.2f} seconds")

# Using a column directly
start = time.time()
large_df['A']
print(f"Time for direct column access: {time.time() - start:.2f} seconds")
```

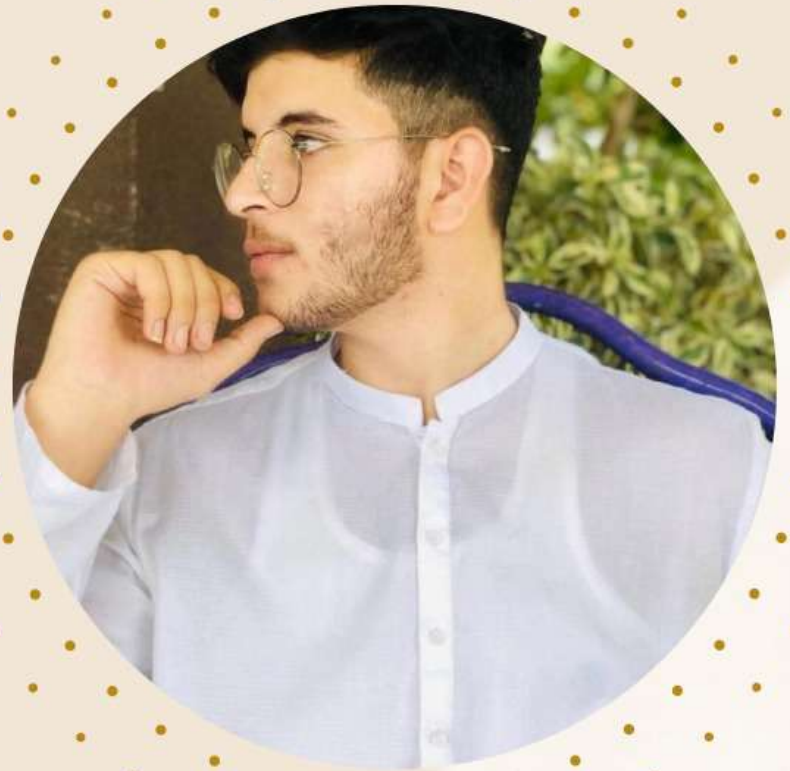
Swipe next ⟶

Additional Resources

For further learning about Pandas and DataFrame operations, consider exploring these resources:

1. Official Pandas Documentation: https://pandas.pydata.org/docs/
2. "Python for Data Analysis" by Wes McKinney (creator of Pandas)
3. DataCamp's Pandas Tutorials: https://www.datacamp.com/community/tutorials/pandas-tutorial-dataframe-python
4. Real Python's Pandas Tutorials: https://realpython.com/pandas-dataframe/
5. Kaggle's Pandas Micro-Course: https://www.kaggle.com/learn/pandas

Data scientist
&ML Engineer



Follow For More Data Science Content