Python Pandas Cheat Sheet

This cheat sheet is a companion to the “Complete Python Pandas Data Science Tutorial.” For a more detailed walkthrough, check out the video here.

Creating DataFrames

Creating DataFrames is the foundation of using Pandas. Here’s how to create a simple DataFrame and display its content.

```
import pandas as pd
import numpy as np

# Create a simple DataFrame
df = pd.DataFrame(
    [[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]],
    columns=["A", "B", "C"],
    index=["x", "y", "z", "zz"]
)

# Display the first few rows
df.head()

# Display the last two rows
df.tail(2)
```

Loading Data

Loading data into DataFrames from various file formats is crucial for real-world data analysis.

```
# Load data from CSV
coffee = pd.read_csv('./warmup-data/coffee.csv')
```

```
# Load data from Parquet

results = pd.read_parquet('./data/results.parquet')


# Load data from Excel

olympics_data = pd.read_excel('./data/olympics-data.xlsx', sheet_name="results")
```

Accessing Data

Accessing different parts of the DataFrame allows for flexible data manipulation and inspection.

```
# Access columns

df.columns


# Access index

df.index.tolist()


# General info about the DataFrame

df.info()


# Statistical summary

df.describe()


# Number of unique values in each column

df.nunique()


# Access unique values in a column

df['A'].unique()


# Shape and size of DataFrame

df.shape

df.size
```

Filtering Data

Filtering data is essential for extracting relevant subsets based on conditions.

```
# Filter rows based on conditions
bios.loc[bios["height_cm"] > 215]

# Multiple conditions
bios[(bios['height_cm'] > 215) & (bios['born_country']=='USA')]

# Filter by string conditions
bios[bios['name'].str.contains("keith", case=False)]

# Regex filters
bios[bios['name'].str.contains(r'^[AEIOUaeiou]', na=False)]
```

Adding/Removing Columns

Adding and removing columns is important for maintaining and analyzing relevant data.

```
# Add a new column
coffee['price'] = 4.99

# Conditional column
coffee['new_price'] = np.where(coffee['Coffee Type']=='Espresso', 3.99, 5.99)

# Remove a column
coffee.drop(columns=['price'], inplace=True)

# Rename columns
coffee.rename(columns={'new_price': 'price'}, inplace=True)

# Create new columns from existing ones
coffee['revenue'] = coffee['Units Sold'] * coffee['price']
```

Merging and Concatenating Data

Merging and concatenating DataFrames is useful for combining different datasets for comprehensive analysis.

```
# Merge DataFrames
nocs = pd.read_csv('./data/noc_regions.csv')
bios_new = pd.merge(bios, nocs, left_on='born_country', right_on='NOC', how='left')

# Concatenate DataFrames
usa = bios[bios['born_country']=='USA'].copy()
gbr = bios[bios['born_country']=='GBR'].copy()
new_df = pd.concat([usa, gbr])
```

Handling Null Values

Handling null values is essential to ensure the integrity of data analysis.

```
# Fill NaNs with a specific value
coffee['Units Sold'].fillna(0, inplace=True)

# Interpolate missing values
coffee['Units Sold'].interpolate(inplace=True)

# Drop rows with NaNs
coffee.dropna(subset=['Units Sold'], inplace=True)
```

Aggregating Data

Aggregation functions like value counts and group by help in summarizing data efficiently.

```
# Value counts
bios['born_city'].value_counts()
```

```
# Group by and aggregation
coffee.groupby(['Coffee Type'])['Units Sold'].sum()
coffee.groupby(['Coffee Type'])['Units Sold'].mean()

# Pivot table
pivot = coffee.pivot(columns='Coffee Type', index='Day', values='revenue')
```

Advanced Functionality

Advanced functionalities such as rolling calculations, rankings, and shifts can provide deeper insights.

```
# Cumulative sum
coffee['cumsum'] = coffee['Units Sold'].cumsum()

# Rolling window
latte = coffee[coffee['Coffee Type']=="Latte"].copy()
latte['3day'] = latte['Units Sold'].rolling(3).sum()

# Rank
bios['height_rank'] = bios['height_cm'].rank(ascending=False)

# Shift
coffee['yesterday_revenue'] = coffee['revenue'].shift(1)
```

New Functionality

The PyArrow backend offers optimized performance for certain operations, particularly string operations.

```
# PyArrow backend
results_arrow = pd.read_csv('./data/results.csv', engine='pyarrow', dtype_backend='pyarrow')
results_arrow.info()
```