Data Operations with NumPy and Pandas

Akhilesh Joshi [02FE22BCS013]

March 6, 2025

Abstract

This document provides practical examples of data operations using NumPy and Pandas libraries for data analysis in Python. Each code snippet is accompanied by explanations to help understand the concepts and functionality. These examples cover data loading, cleaning, transformation, analysis, and visualization preparation.

Contents

1 NumPy for Data Operations

1.1 Data Creation and Basic Operations

Python Code

```
# Import NumPy
import numpy as np

# Creating arrays
array1d = np.array([1, 2, 3, 4, 5])  # 1D array
array2d = np.array([[1, 2, 3], [4, 5, 6]])  # 2D array

# Generate arrays with patterns
zeros = np.zeros((3, 4))  # 3x4 array filled with zeros
ones = np.ones((2, 2))  # 2x2 array filled with ones
seq = np.arange(0, 10, 2)  # [0, 2, 4, 6, 8]
lin = np.linspace(0, 1, 5)  # 5 evenly spaced values from 0 to 1

# Random data generation
uniform = np.random.rand(3, 3)  # 3x3 array with random values
    [0,1)
normal = np.random.randn(100)  # 100 samples from standard normal
    distribution
integers = np.random.randint(1, 100, size=10)  # 10 random
    integers from 1 to 99
```

1.2 Array Manipulation and Reshaping

Python Code

```
# Create sample data
arr = np.arange(12)  # [0, 1, 2, ..., 11]

# Reshape to 2D array
arr_2d = arr.reshape(3, 4)  # 3 rows, 4 columns
arr_2d_alt = arr.reshape(4, -1)  # 4 rows, columns calculated
    automatically

# Flatten and ravel
flat = arr_2d.flatten()  # Returns a copy
rav = arr_2d.ravel()  # Returns a view when possible (more
    efficient)

# Transpose
transposed = arr_2d.T  # Swap rows and columns

# Stacking arrays
```

```
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
vertical = np.vstack((a, b))  # Stack vertically
horizontal = np.hstack((a, b))  # Stack horizontally
tiled = np.tile(a, 3)  # Repeat array: [1, 2, 3, 1, 2, 3, 1, 2, 3]
```

1.3 Indexing and Slicing

Python Code

```
# Create a 2D array
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])

# Basic indexing
element = arr[1, 2]  # Value at row 1, column 2 (7)
row = arr[0]  # First row [1, 2, 3, 4]
column = arr[:, 1]  # Second column [2, 6, 10]

# Slicing
slice_rows = arr[0:2]  # First two rows
slice_block = arr[0:2, 1:3]  # Block from rows 0-1, columns 1-2

# Boolean indexing
mask = arr > 6  # Boolean mask where values are > 6
filtered = arr[mask]  # 1D array of values > 6

# Fancy indexing
row_indices = np.array([0, 2])  # Select rows 0 and 2
col_indices = np.array([1, 3])  # Select columns 1 and 3
selected = arr[row_indices[:, np.newaxis], col_indices]  #
    Combination of rows and columns
```

1.4 Mathematical Operations

Python Code

```
# Create arrays
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

# Basic operations
addition = a + b  # [5, 7, 9]
subtraction = b - a  # [3, 3, 3]
multiplication = a * b  # [4, 10, 18]
division = b / a  # [4.0, 2.5, 2.0]

# More operations
```

```
squared = a ** 2  # [1, 4, 9]
sqrt = np.sqrt(a)  # [1.0, 1.41, 1.73]
exp = np.exp(a)  # [e^1, e^2, e^3]
log = np.log(a)  # [ln(1), ln(2), ln(3)]

# Aggregation
total = np.sum(a)  # 6
maximum = np.max(a)  # 3
minimum = np.min(a)  # 1
mean_val = np.mean(a)  # 2.0
median_val = np.median(a)  # 2.0
std_dev = np.std(a)  # Standard deviation
```

1.5 Data Analysis with NumPy

Python Code

```
# Generate sample data
np.random.seed(42)
data = np.random.normal(loc=50, scale=10, size=1000)

# Basic statistics
summary = {
    'mean': np.mean(data),
    'median': np.median(data),
    'std': np.std(data),
    'min': np.min(data),
    'max': np.max(data)
}

# Percentiles
percentiles = np.percentile(data, [25, 50, 75])  # Q1, Q2, Q3
iqr = percentiles[2] - percentiles[0]  # Interquartile Range

# Identify outliers (using 1.5 * IQR rule)
lower_bound = percentiles[0] - 1.5 * iqr
upper_bound = percentiles[2] + 1.5 * iqr
outliers = data[(data < lower_bound) | (data > upper_bound)]

# Data binning
hist, bin_edges = np.histogram(data, bins=10)  # Frequency
   histogram
```

2 Pandas for Data Operations

2.1 Series and DataFrames

Python Code

```
# Import Pandas
import pandas as pd
import numpy as np

# Create Series (1D labeled array)
s1 = pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])
s2 = pd.Series({'a': 10, 'b': 20, 'c': 30})  # From dictionary

# Create DataFrame (2D labeled data structure)
df1 = pd.DataFrame({
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Age': [25, 30, 35, 40],
    'City': ['New York', 'London', 'Paris', 'Tokyo']
})

# Create DataFrame from NumPy array
arr = np.random.randn(4, 3)  # Random data
df2 = pd.DataFrame(
    arr,
    columns=['A', 'B', 'C'],
    index=['Row1', 'Row2', 'Row3', 'Row4']
)
```

2.2 Data Loading and Inspection

Python Code

```
# Sample datasets from built-in data
import seaborn as sns
titanic = sns.load_dataset('titanic')  # Load Titanic dataset

# Inspect data
head = titanic.head()  # First 5 rows
tail = titanic.tail(3)  # Last 3 rows
info = titanic.info()  # Column info, non-null counts, and data
   types
stats = titanic.describe()  # Summary statistics for numeric
   columns

# Structure information
shape = titanic.shape  # (rows, columns)
columns = titanic.columns  # Column names
dtypes = titanic.dtypes  # Data types of each column
value_counts = titanic['survived'].value_counts()  # Count of each
    value
```

2.3 Data Selection and Filtering

Python Code

```
# Using the Titanic dataset
titanic = sns.load_dataset('titanic')

# Column selection
ages = titanic['age']  # Select one column as Series
subset = titanic[['name', 'age', 'survived']]  # Select multiple
    columns

# Row selection
first_row = titanic.iloc[0]  # First row as Series
rows_subset = titanic.iloc[10:15]  # Rows 10-14
specific_cells = titanic.iloc[0:5, 2:4]  # Rows 0-4, columns 2-3

# Row selection by label
row_labels = titanic.loc[titanic.index[5:10]]  # Rows 5-9

# Boolean filtering
survivors = titanic[titanic['survived'] == 1]  # Only survivors
adult_males = titanic[(titanic['age'] >= 18) & (titanic['sex'] ==
    'male')]  # Adult males

# String filtering
english = titanic[titanic['embark_town'].str.contains('South', na=
    False)]  # Contains 'South'

# Top/bottom rows by value
richest = titanic.nlargest(5, 'fare')  # 5 highest fares
```

2.4 Data Cleaning and Preprocessing

Python Code

```
# Using the Titanic dataset
titanic = sns.load_dataset('titanic')

# Check missing values
missing = titanic.isna().sum()  # Count missing values per column
missing_percent = titanic.isna().mean() * 100  # Percentage
    missing

# Handle missing values
titanic_clean = titanic.copy()
# Fill missing values with mean for age
```

```
titanic_clean['age'] = titanic_clean['age'].fillna(titanic_clean['
    age'].mean())
# Fill missing values with mode for embarked
titanic_clean['embarked'] = titanic_clean['embarked'].fillna(
    titanic_clean['embarked'].mode()[0])
# Drop rows with any remaining missing values
titanic_clean = titanic_clean.dropna()

# Duplicate detection and removal
duplicates = titanic.duplicated().sum()  # Count duplicates
titanic_unique = titanic.drop_duplicates()  # Remove duplicates

# Data type conversion
titanic_clean['survived'] = titanic_clean['survived'].astype(bool)
     # Convert to boolean
```

2.5 Feature Creation and Transformation

Python Code

```
# Using the Titanic dataset
titanic = sns.load_dataset('titanic')

# Create new columns
titanic['family_size'] = titanic['sibsp'] + titanic['parch'] + 1
    # Family size including self
titanic['is_alone'] = (titanic['family_size'] == 1)  # Boolean:
    traveling alone

# Binning numeric data
titanic['age_group'] = pd.cut(
    titanic['age'],
    bins=[0, 12, 18, 35, 60, float('inf')],
    labels=['Child', 'Teen', 'Young Adult', 'Adult', 'Senior']
)

# One-hot encoding categorical variables
embarked_dummies = pd.get_dummies(titanic['embarked'], prefix='
    embarked')
titanic = pd.concat([titanic, embarked_dummies], axis=1)

# Normalize numeric columns
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
titanic['fare_normalized'] = scaler.fit_transform(titanic[['fare'
    ]])
```

2.6 Grouping and Aggregation

Python Code

```
# Using the Titanic dataset
titanic = sns.load_dataset('titanic')

# Basic groupby operations
class_stats = titanic.groupby('pclass')['fare', 'age'].mean()  #
    Mean by class
class_survival = titanic.groupby('pclass')['survived'].mean()  #
    Survival rate by class

# Multiple aggregations
class_sex_stats = titanic.groupby(['pclass', 'sex']).agg({
    'survived': 'mean',  # Survival rate
    'fare': ['mean', 'median', 'count'],  # Multiple metrics for
    fare
    'age': ['min', 'max', 'mean']  # Age statistics
})

# Transformation within groups
titanic['fare_vs_class_mean'] = titanic['fare'] / titanic.groupby(
    'pclass')['fare'].transform('mean')

# Filter groups
large_groups = titanic.groupby('pclass').filter(lambda x: x['fare'
    ].mean() > 30)
```

2.7 Merging, Joining, and Concatenating Data

Python Code

```
# Sample dataframes
passengers = pd.DataFrame({
    'passenger_id': [1, 2, 3, 4, 5],
    'name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],
    'age': [25, 30, 35, 40, 45]
})

tickets = pd.DataFrame({
    'passenger_id': [1, 2, 3, 6, 7],
    'ticket_number': ['A123', 'B456', 'C789', 'D012', 'E345'],
    'fare': [50, 75, 100, 125, 150]
})

# Merge (similar to SQL join)
```

```
inner_join = pd.merge(passengers, tickets, on='passenger_id')  #
    Inner join
left_join = pd.merge(passengers, tickets, on='passenger_id', how='
    left')  # Left join
right_join = pd.merge(passengers, tickets, on='passenger_id', how=
    'right')  # Right join
outer_join = pd.merge(passengers, tickets, on='passenger_id', how=
    'outer')  # Full outer join

# Concatenation
north_data = pd.DataFrame({'city': ['New York', 'Boston'], 'temp':
     [15, 12]})
south_data = pd.DataFrame({'city': ['Miami', 'Dallas'], 'temp':
    [28, 25]})
all_data = pd.concat([north_data, south_data])  # Stack vertically
all_data_h = pd.concat([north_data, south_data], axis=1)  # Stack
    horizontally
```

2.8 Time Series Operations

Python Code

```
# Create time series data
dates = pd.date_range('2023-01-01', periods=100, freq='D')  # 100
    days
ts = pd.Series(np.random.normal(0, 1, 100), index=dates)  # Random
     values

# Date arithmetic and filtering
next_week = ts['2023-01-08':'2023-01-14']  # Slice by date
jan_data = ts[ts.index.month == 1]  # All January data
weekends = ts[ts.index.dayofweek >= 5]  # Weekend data

# Resampling and frequency conversion
monthly_mean = ts.resample('M').mean()  # Monthly average
weekly_sum = ts.resample('W').sum()  # Weekly sum
rolling_7d = ts.rolling(window=7).mean()  # 7-day rolling average

# Time zone handling
ts_pacific = ts.tz_localize('US/Pacific')  # Add time zone
ts_eastern = ts_pacific.tz_convert('US/Eastern')  # Convert time
    zone
```

2.9 Visualization with Pandas and Matplotlib

Python Code

```
# Import visualization libraries
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

# Load dataset
titanic = sns.load_dataset('titanic')

# Simple plots with pandas
titanic['age'].plot(kind='hist', bins=20, title='Age Distribution'
    )
titanic.groupby('pclass')['survived'].mean().plot(kind='bar',
    title='Survival Rate by Class')

# Scatter plot
plt.figure(figsize=(10, 6))
plt.scatter(titanic['age'], titanic['fare'], c=titanic['survived'
    ], alpha=0.6)
plt.xlabel('Age')
plt.ylabel('Fare')
plt.title('Age vs. Fare (color=survived)')
plt.colorbar(label='Survived')

# Multiple plots with subplots
fig, axes = plt.subplots(1, 2, figsize=(12, 5))
# Age distribution by sex
titanic.groupby('sex')['age'].plot(kind='kde', ax=axes[0])
axes[0].set_title('Age Distribution by Sex')
axes[0].legend(['Female', 'Male'])
# Survival count by embarkation point
sns.countplot(x='embarked', hue='survived', data=titanic, ax=axes
    [1])
axes[1].set_title('Survival by Embarkation Point')
plt.tight_layout()

# Heat map of correlation
plt.figure(figsize=(10, 8))
numeric_data = titanic.select_dtypes(include=['float64', 'int64'])
correlation = numeric_data.corr()
sns.heatmap(correlation, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Between Numeric Features')
```

2.10 Advanced Pandas Operations

Python Code

```
# Using the Titanic dataset
titanic = sns.load_dataset('titanic')
```

```
# Apply custom functions
def get_title(name):
    """Extract title from passenger name"""
    title = name.split(',')[1].split('.')[0].strip()
    return title

titanic['title'] = titanic['name'].apply(get_title)

# Pivot tables
survival_pivot = titanic.pivot_table(
    values='survived',
    index='pclass',
    columns='sex',
    aggfunc='mean'
)

# Multi-level indexing
grouped = titanic.groupby(['pclass', 'sex'])
multi_index_df = grouped.agg({
    'survived': 'mean',
    'age': 'mean',
    'fare': 'mean'
})

# Stack and unstack
stacked = multi_index_df.stack()  # Pivot a level of column labels
unstacked = stacked.unstack(0)  # Pivot a level of index labels

# Value counts with normalization
class_distribution = titanic['pclass'].value_counts(normalize=True
   )  # As proportions

# Custom sorting
sorted_class = titanic.groupby('pclass')['survived'].mean().
   sort_values(ascending=False)
```

3 Real-world Data Analysis Example

Python Code

```
# Complete data analysis example
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# 1. Load and explore data
titanic = sns.load_dataset('titanic')
print("Dataset shape:", titanic.shape)
print("\nMissing values per column:")
print(titanic.isna().sum())

# 2. Clean data
titanic_clean = titanic.copy()
# Fill missing ages with median by passenger class and sex
titanic_clean['age'] = titanic_clean.groupby(['pclass', 'sex'])['
    age'].transform(
     lambda x: x.fillna(x.median())
)
# Fill missing embarked with mode
most_common_port = titanic_clean['embarked'].mode()[0]
titanic_clean['embarked'] = titanic_clean['embarked'].fillna(
    most_common_port)
# Drop cabin (too many missing) and drop rows with remaining NAs
titanic_clean = titanic_clean.drop('cabin', axis=1).dropna()

# 3. Create new features
# Family size
titanic_clean['family_size'] = titanic_clean['sibsp'] +
    titanic_clean['parch'] + 1
# Create age groups
titanic_clean['age_group'] = pd.cut(
     titanic_clean['age'],
     bins=[0, 12, 18, 35, 60, float('inf')],
     labels=['Child', 'Teen', 'Young Adult', 'Adult', 'Senior']
)
# Fare per person
titanic_clean['fare_pp'] = titanic_clean['fare'] / titanic_clean['
    family_size']
# One-hot encode categorical features
categorical_features = ['sex', 'embarked', 'age_group']
titanic_encoded = pd.get_dummies(titanic_clean, columns=
    categorical_features, drop_first=True)

# 4. Analyze correlations with survival
corr = titanic_encoded.corr()['survived'].sort_values(ascending=
    False)
print("\nCorrelations with survival:")
print(corr)

# 5. Calculate survival rates by different groups
survival_by_class = titanic_clean.groupby('pclass')['survived'].
    mean()
survival_by_sex = titanic_clean.groupby('sex')['survived'].mean()
```

```
survival_by_age_group = titanic_clean.groupby('age_group')['
    survived'].mean()

# 6. Visualize key findings
plt.figure(figsize=(15, 10))

# Survival by sex and class
plt.subplot(2, 2, 1)
sns.barplot(x='pclass', y='survived', hue='sex', data=
    titanic_clean)
plt.title('Survival Rate by Class and Sex')

# Age distribution by survival
plt.subplot(2, 2, 2)
sns.kdeplot(
    data=titanic_clean, x='age', hue='survived',
    multiple='stack', palette=['red', 'green']
)
plt.title('Age Distribution by Survival Status')

# Fare vs Survival
plt.subplot(2, 2, 3)
sns.boxplot(x='pclass', y='fare_pp', hue='survived', data=
    titanic_clean)
plt.title('Fare per Person by Class and Survival')
plt.yscale('log')

# Family size vs Survival
plt.subplot(2, 2, 4)
family_survival = titanic_clean.groupby('family_size')['survived'
    ].mean().reset_index()
sns.barplot(x='family_size', y='survived', data=family_survival)
plt.title('Survival Rate by Family Size')

plt.tight_layout()
plt.savefig('titanic_analysis.png')

# 7. Statistical testing
from scipy import stats
# Test if survival differs significantly between males and females
male_survival = titanic_clean[titanic_clean['sex'] == 'male']['
    survived']
female_survival = titanic_clean[titanic_clean['sex'] == 'female'][
    'survived']
t_stat, p_val = stats.ttest_ind(female_survival, male_survival)
print(f"\nT-test for survival difference by sex: p-value = {p_val
    :.10f}")
```

4 Conclusion

NumPy and Pandas are essential tools for data manipulation and analysis in Python. NumPy provides the foundation with efficient array operations, while Pandas builds on this with intuitive data structures like Series and DataFrames. Together, they offer a comprehensive toolkit for:

- Loading and exploring data from various sources
- Cleaning and preprocessing data for analysis
- Transforming and manipulating data structures
- Performing statistical analysis and aggregations
- Creating meaningful visualizations
- Building pipelines for data science and machine learning

Mastering these libraries is essential for anyone working with data analysis or data science in Python. The examples provided in this document serve as a starting point for understanding the capabilities of these powerful libraries.