# Detailed Document on Pandas DataFrame Operations

---

## Introduction to Pandas DataFrame #

Pandas is a powerful Python library for data analysis and manipulation. A DataFrame is a two-dimensional, size-mutable, and heterogeneous data structure, similar to a table in a relational database or an Excel spreadsheet.

---

## Creating a DataFrame #

```
import pandas as pd

# Creating a DataFrame from a dictionary
data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25, 30, 35],
    'Salary': [50000, 60000, 70000]
}

df = pd.DataFrame(data)
print(df)
```

---

## Basic Operations #

### 1. Viewing Data #

- `head(n)`: View the first `n` rows (default: 5).
- `tail(n)`: View the last `n` rows (default: 5).
- `info()`: Summary of the DataFrame.
- `describe()`: Statistical summary of numerical columns.

```
print(df.head(2))
print(df.tail(2))
print(df.info())
print(df.describe())
```

### 2. Accessing Data #

- Column selection: `df['column_name']` or `df.column_name`
- Row selection: `df.loc[index]` (label-based) or `df.iloc[index]` (integer-based)

```
# Accessing a column
print(df['Name'])

# Accessing a row by label
print(df.loc[1])

# Accessing a row by index
print(df.iloc[1])
```

---

## Data Manipulation #

### 1. Adding Columns #

```
df['Department'] = ['HR', 'Finance', 'IT']
print(df)
```

### 2. Dropping Columns or Rows #

- `drop()`: Remove specific rows or columns.

```
# Dropping a column
df = df.drop('Department', axis=1)

# Dropping a row
df = df.drop(1, axis=0)
```

### 3. Renaming Columns #

```
df.rename(columns={'Name': 'Employee Name'}, inplace=True)
print(df)
```

### 4. Filtering Data #

```
# Filter rows where Age > 28
filtered_df = df[df['Age'] > 28]
print(filtered_df)
```

### 5. Sorting Data #

```
# Sort by Age
sorted_df = df.sort_values(by='Age', ascending=True)
print(sorted_df)
```

## Aggregation and Grouping #

### 1. Aggregation Functions #

- `sum()`, `mean()`, `min()`, `max()`, etc.

```
print(df['Salary'].sum())
print(df['Age'].mean())
```

### 2. Grouping Data #

```
grouped = df.groupby('Department').mean()
print(grouped)
```

## Handling Missing Data #

### 1. Detecting Missing Values #

- `isnull()`: Check for missing values.
- `notnull()`: Check for non-missing values.

```
print(df.isnull())
```

### 2. Filling Missing Values #

```
# Fill missing values with a specific value
df['Age'] = df['Age'].fillna(30)
```

### 3. Dropping Missing Values #

```
# Drop rows with missing values
df = df.dropna()
```

---

## Merging, Joining, and Concatenation #

### 1. Merging #

```
data2 = {
    'Name': ['Alice', 'Bob'],
    'City': ['New York', 'Los Angeles']
}
df2 = pd.DataFrame(data2)
merged_df = pd.merge(df, df2, on='Name')
print(merged_df)
```

### 2. Concatenation #

```
concat_df = pd.concat([df, df2], axis=0)
print(concat_df)
```

### 3. Joining #

```
joined_df = df.set_index('Name').join(df2.set_index('Name'))
print(joined_df)
```

---

## Advanced Operations #

### 1. Applying Functions #

```
# Apply a lambda function to a column
df['Double Salary'] = df['Salary'].apply(lambda x: x * 2)
print(df)
```

### 2. Pivot Tables #

```
pivot = df.pivot_table(values='Salary', index='Department', aggfunc='mean')
print(pivot)
```

### 3. Working with Dates #

```
df['Hire Date'] = pd.to_datetime(['2020-01-01', '2019-05-20', '2021-07-15'])
print(df)
print(df['Hire Date'].dt.year)
```

---

## Saving and Loading Data #

### 1. Saving to a File #

```
# Save to CSV
df.to_csv('data.csv', index=False)

# Save to Excel
df.to_excel('data.xlsx', index=False)
```

### 2. Loading from a File #

```
# Load from CSV
new_df = pd.read_csv('data.csv')

# Load from Excel
new_df = pd.read_excel('data.xlsx')
```

---

## Conclusion #

Pandas DataFrame provides a versatile and efficient way to handle and analyze structured data. Mastering these operations will significantly enhance your data analysis workflow.