Unit-V

Dr.J.Ramesh Babu

- Unit-V: Data Analysis
- Introduction to NumPy: arrays and basic operations; Introduction to Pandas: Series and Data Frames; Basic data manipulation and summary statistics; Introduction to Matplotlib: plotting line and bar graphs; Exploratory Data Analysis (EDA); Solve mesh or nodal circuit equations using NumPy arrays. Plot hourly load profiles of electrical demand using Matplotlib.

Introduction to NumPy: Arrays and Basic Operations

NumPy (Numerical Python) is a foundational library for numerical computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently.

Key Concepts:

•ndarray (N-dimensional array):

- The core data structure in NumPy.
- Homogeneous (all elements must be of the same data type).
- Fixed size upon creation.

```
import numpy as np

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

Basic Operations:

- **Element-wise arithmetic**: +, -, *, /
- **Broadcasting**: operations between arrays of different shapes.
- **Mathematical functions**: np.sin(), np.exp(), np.sqrt()
- **Array attributes**: .shape, .dtype, .size
- **Indexing and slicing**: similar to Python lists but extended to multiple dimensions.

```
c = a + 1          # Adds 1 to every element
d = a * b          # Element-wise multiplication (if shapes compatible)
```

Introduction to Pandas: Series and DataFrames

Pandas is a powerful data manipulation and analysis library built on top of NumPy. It provides high-level data structures ideal for working with **tabular data**.

Core Data Structures:

1.Series:

1. A one-dimensional labeled array.
2. Can hold any data type (int, float, string, etc.).
3. Similar to a column in a spreadsheet.

```
import pandas as pd

s = pd.Series([10, 20, 30], index=['a', 'b', 'c'])
```

2.DataFrame:

2. A two-dimensional labeled data structure with columns of potentially different types.
3. Think of it as a **spreadsheet or SQL table**.

```
df = pd.DataFrame({
    'Name': ['jrb', 'charith', 'mouhith'],
    'Age': [41, 14, 08],
    'City': ['HYD', 'KMM', 'Una']
})
```

Key Features:

- Automatic and explicit **indexing**.
- Handles **missing data** (NaN) gracefully.
- Supports **reading/writing** from CSV, Excel, JSON, SQL, etc.

Basic Data Manipulation and Summary Statistics

Pandas excels at **data wrangling**—cleaning, filtering, and transforming data.

Common Operations:

- **Selection**:
  - df['column'] or df[['col1', 'col2']]
  - df.loc[] (label-based), df.iloc[] (integer-position-based)

Filtering

```
df[df['Age'] > 25]
```

Adding/Modifying Columns:

```
df['Salary'] = [50000, 60000, 70000]
```

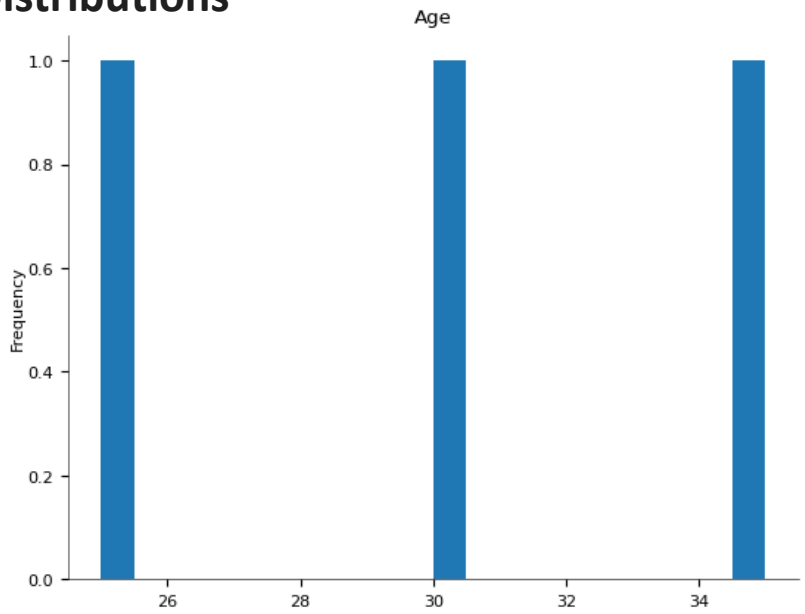
Handling Missing Data

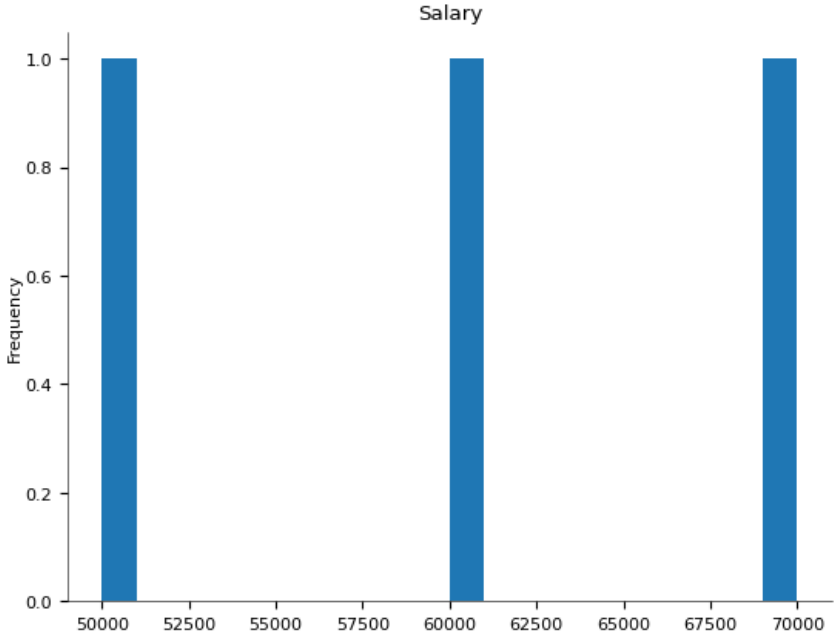
```
df.dropna()          # Remove rows with NaN
df.fillna(0)         # Replace NaN with 0
```

```
df.dropna()          # Remove rows with NaN
df.fillna(0)         # Replace NaN with 0
```

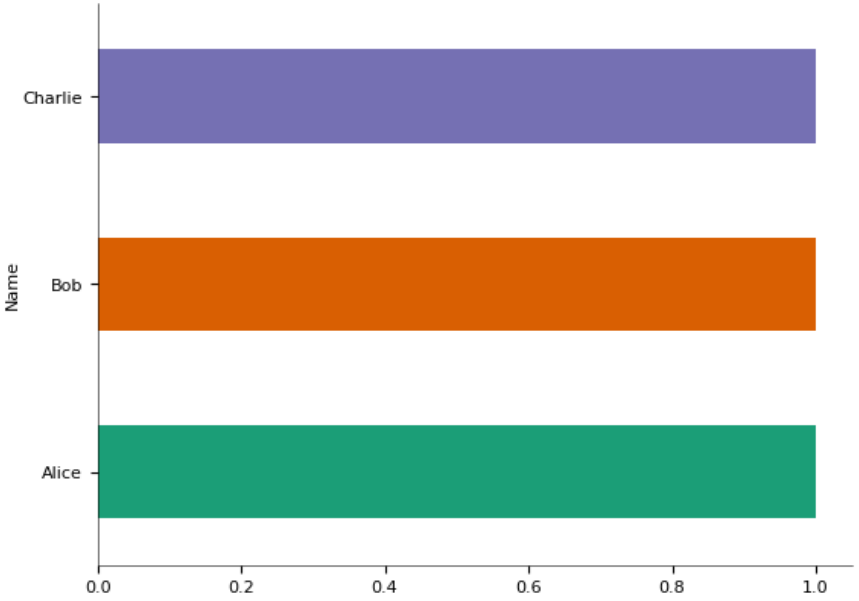
	Name	Age	City	Salary
0	Alice	25	NY	50000
1	Bob	30	LA	60000
2	Charlie	35	Chicago	70000

Distributions

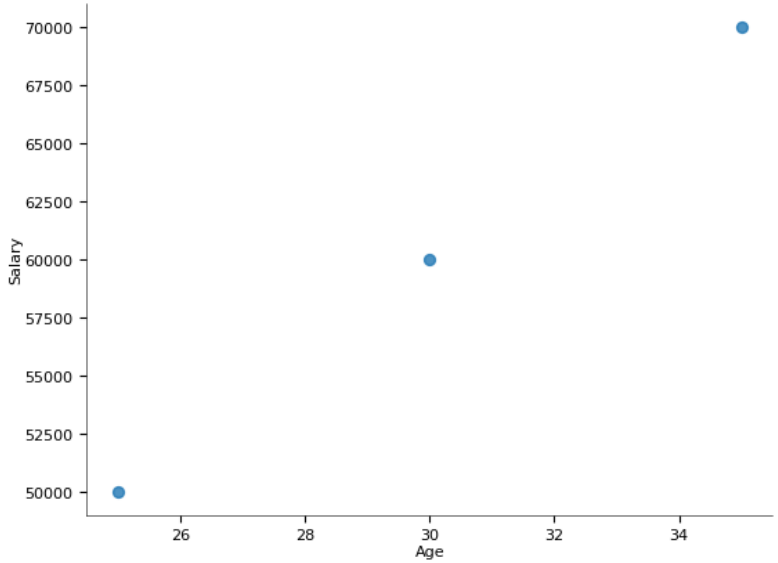




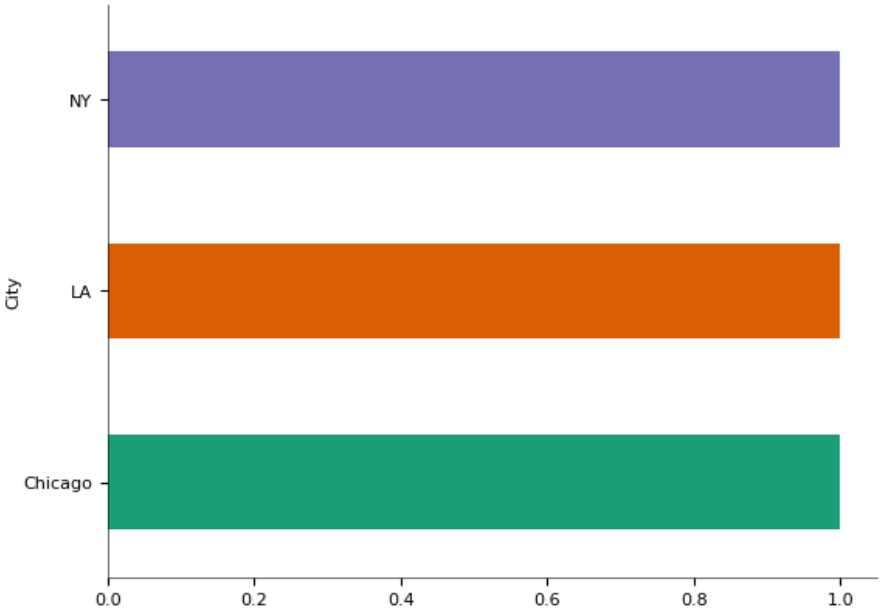
Categorical distributions

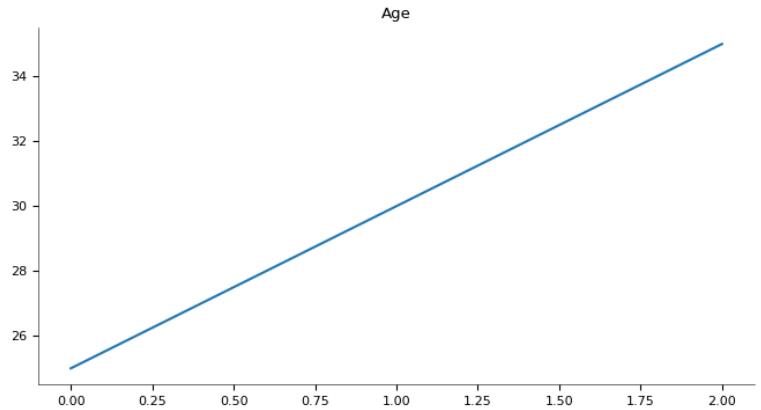


2-d distributions



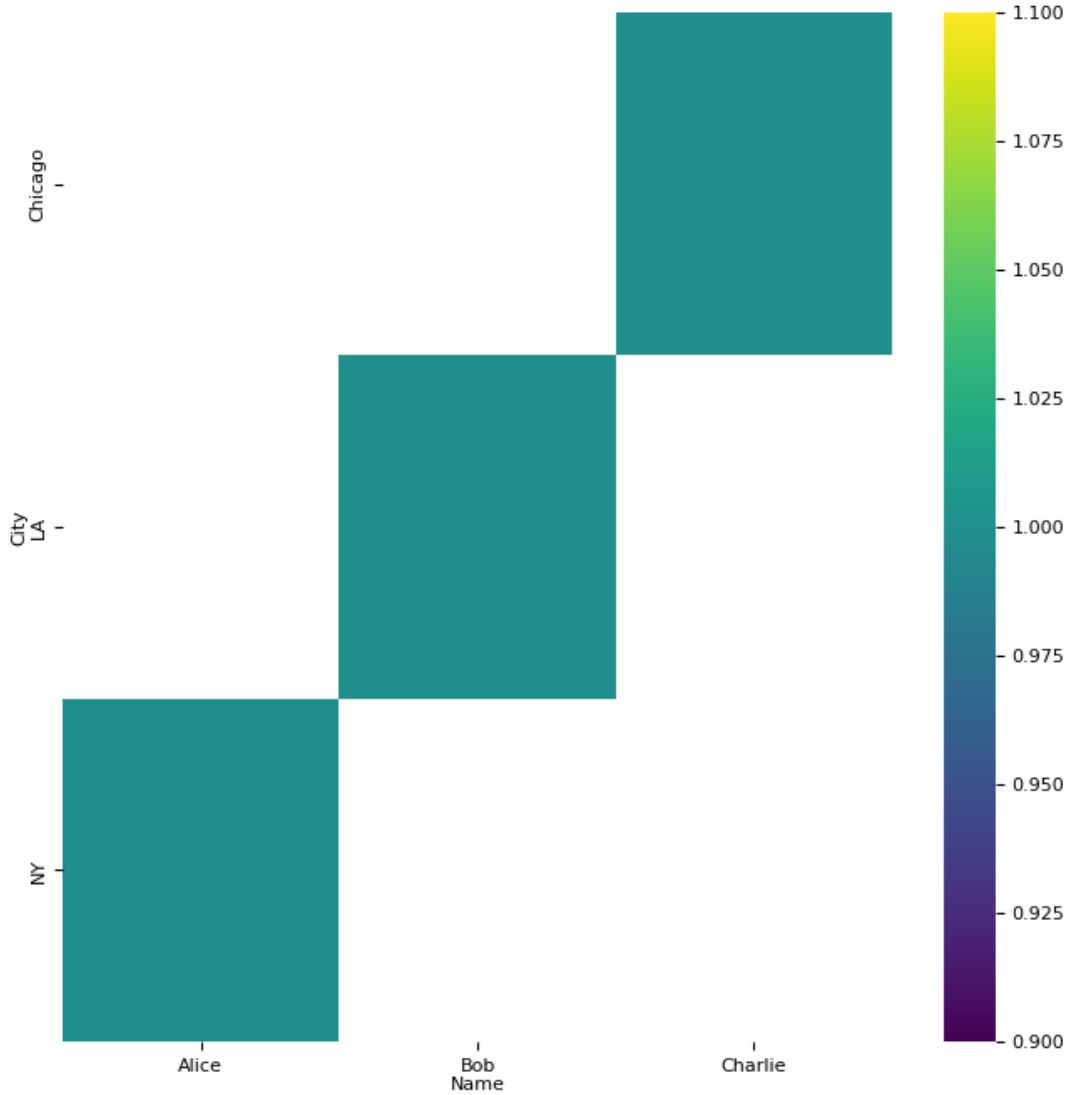
Values







2-d categorical distributions



Summary Statistics:

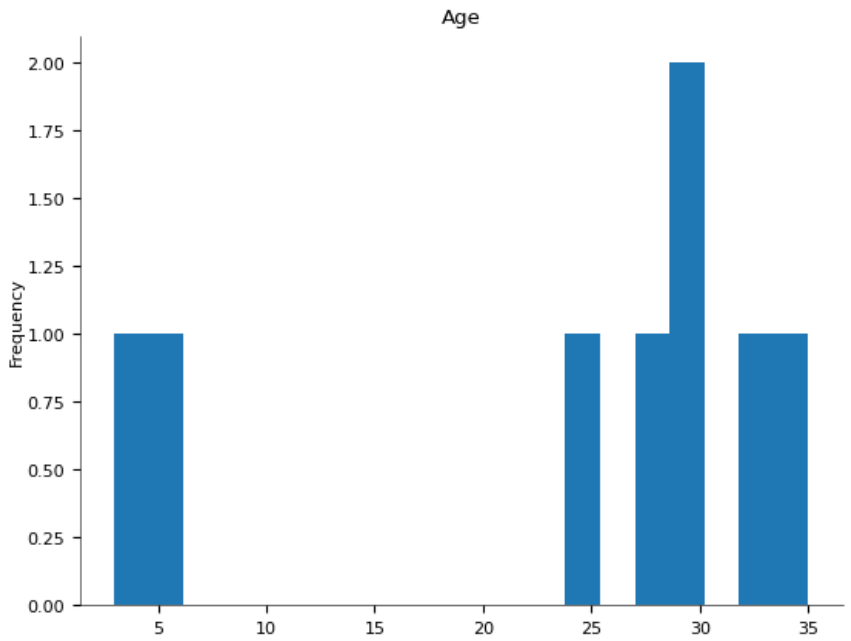
Pandas provides built-in methods for descriptive statistics:

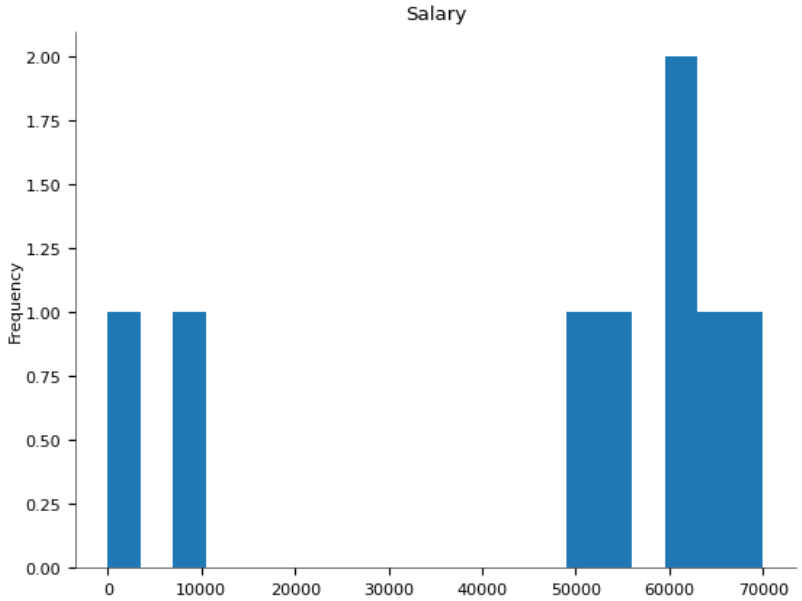
Method	Description
.mean()	Arithmetic mean
.median()	Median
.std()	Standard deviation
.min()" ".max()	Minimum and maximum
.describe()	Summary of all statistics
.value_counts()	Frequency count for categories

```
df.describe()  # Shows count, mean, std, min, 25%, 50%, 75%, max
```

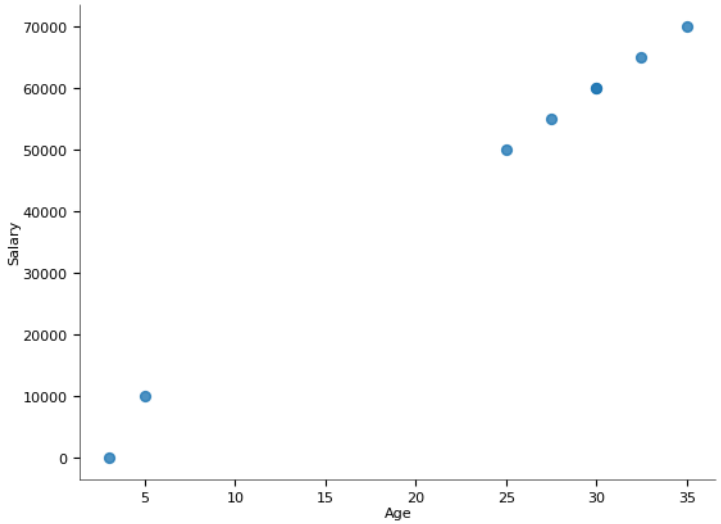
	Age	Salary
count	3.0	3.0
mean	30.0	60000.0
std	5.0	10000.0
min	25.0	50000.0
25%	27.5	55000.0
50%	30.0	60000.0
75%	32.5	65000.0
max	35.0	70000.0

Distributions

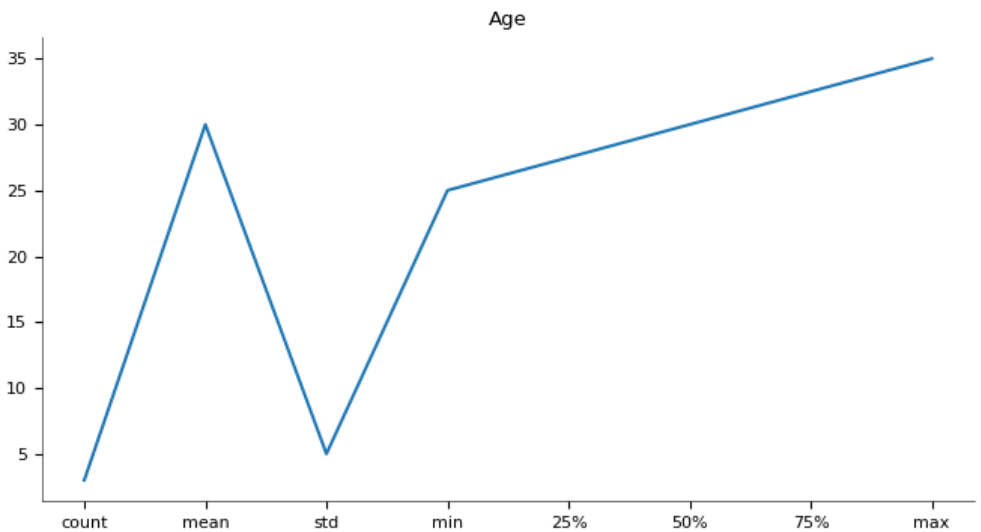




2-d distributions



Values





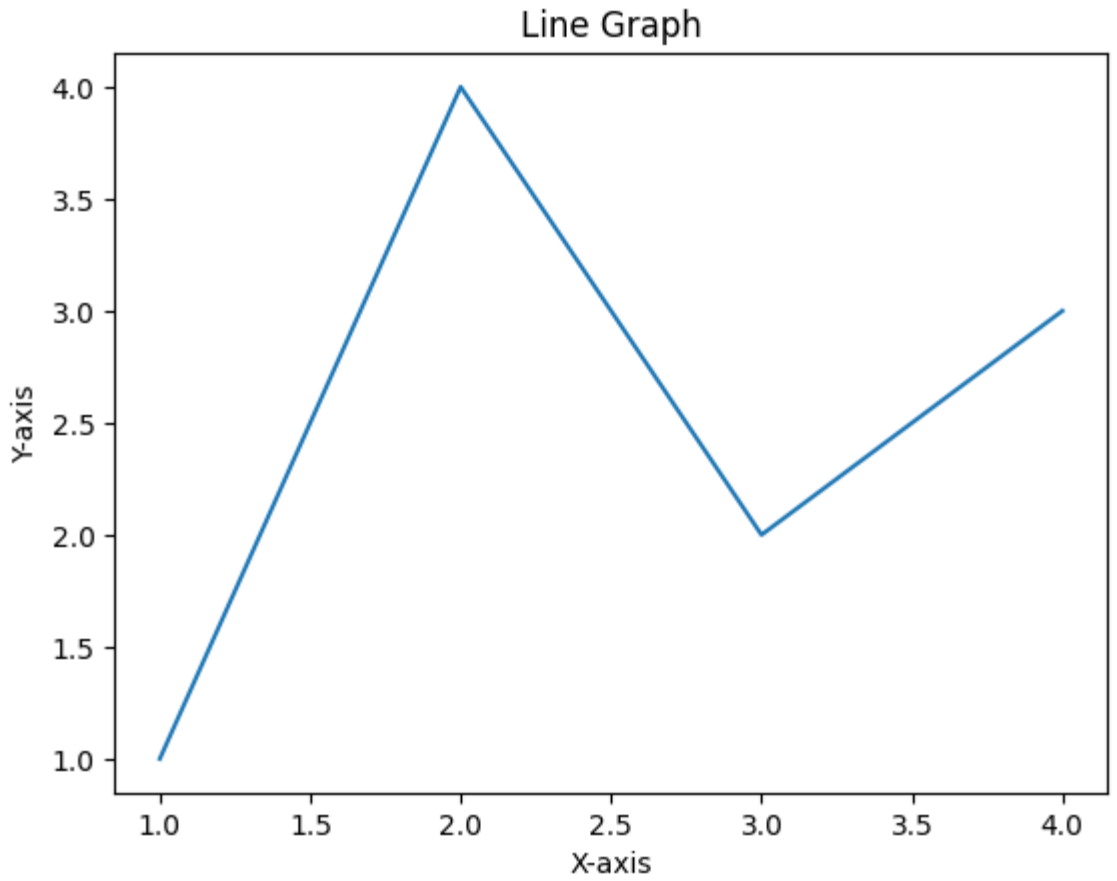
Introduction to Matplotlib: Plotting Line and Bar Graphs

Matplotlib is the most widely used 2D plotting library in Python. It's highly customizable and integrates well with NumPy and Pandas.

Basic Workflow:

```
mport matplotlib.pyplot as pl

# Line plot
x = [1, 2, 3, 4]
y = [1, 4, 2, 3]
plt.plot(x, y)
plt.title("Line Graph")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.show()
```

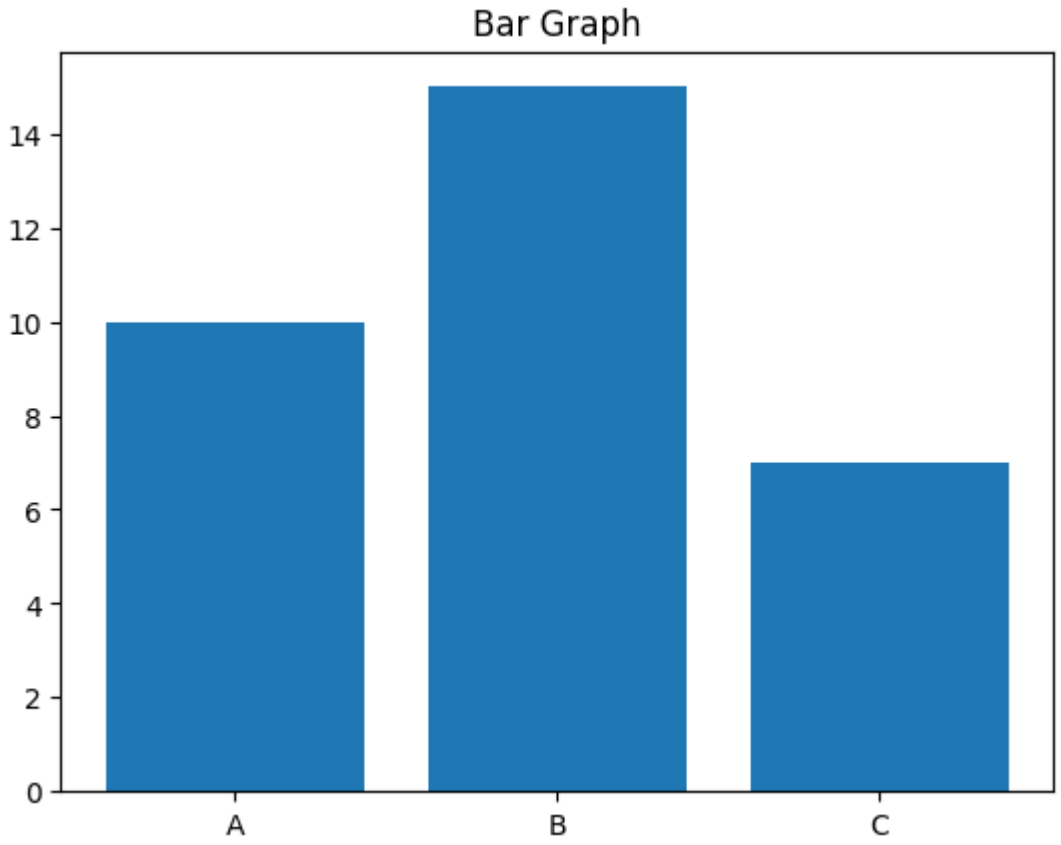


Common Plot Types:

•**Line Plot**: plt.plot() — for trends over time or continuous data.

•**Bar Graph**: plt.bar() — for categorical comparisons.

```
# Bar graph
categories = ['A', 'B', 'C']
values = [10, 15, 7]
plt.bar(categories, values)
plt.title("Bar Graph")
plt.show()
```



Customization:

•Labels, titles, legends, colors, markers, grid lines.

•Multiple subplots using plt.subplot() or fig, ax = plt.subplots().

Exploratory Data Analysis (EDA)

EDA is the process of **understanding a dataset** through summarization and visualization before modeling.

Goals of EDA:

- Discover patterns, anomalies, and relationships.
- Check assumptions (e.g., normality, linearity).
- Identify missing or erroneous data.

Check Steps in EDA:

1.Inspect the data:
- df.head(), df.info(), df.shape

2.for missing values:
- df.isnull().sum()

3.Univariate analysis:
- Histograms, box plots for individual variables.

4.Bivariate/Multivariate analysis:
- Scatter plots, correlation matrices (df.corr())

5.Group-wise analysis:
- df.groupby('Category').mean()

6.Visualization:
- Use Matplotlib or Seaborn for insights.

Application 1: Solving Mesh or Nodal Circuit Equations Using NumPy Arrays

In electrical engineering, **mesh analysis** (loop currents) or **nodal analysis** (node voltages) leads to systems of linear equations:

$Ax = b$

Where:

- **A** = coefficient matrix (from KVL or KCL)
- **x** = unknown vector (currents or voltages)
- **b** = constant vector (voltage sources or current sources)

Using NumPy:

```
import numpy as np

# Example nodal equations:
# 3V1 - V2 = 10
# -V1 + 4V2 = 5

A = np.array([[3, -1],
              [-1, 4]])
b = np.array([10, 5])

# Solve for [V1, V2]
voltages = np.linalg.solve(A, b)
print("Node voltages:", voltages)
```

Node voltages: [4.09090909 2.27272727]

Application 2: Plot Hourly Load Profiles of Electrical Demand Using Matplotlib

Load profile: A graph showing electricity demand (in kW or MW) over time (e.g., 24 hours).

Steps:

1.Prepare data:

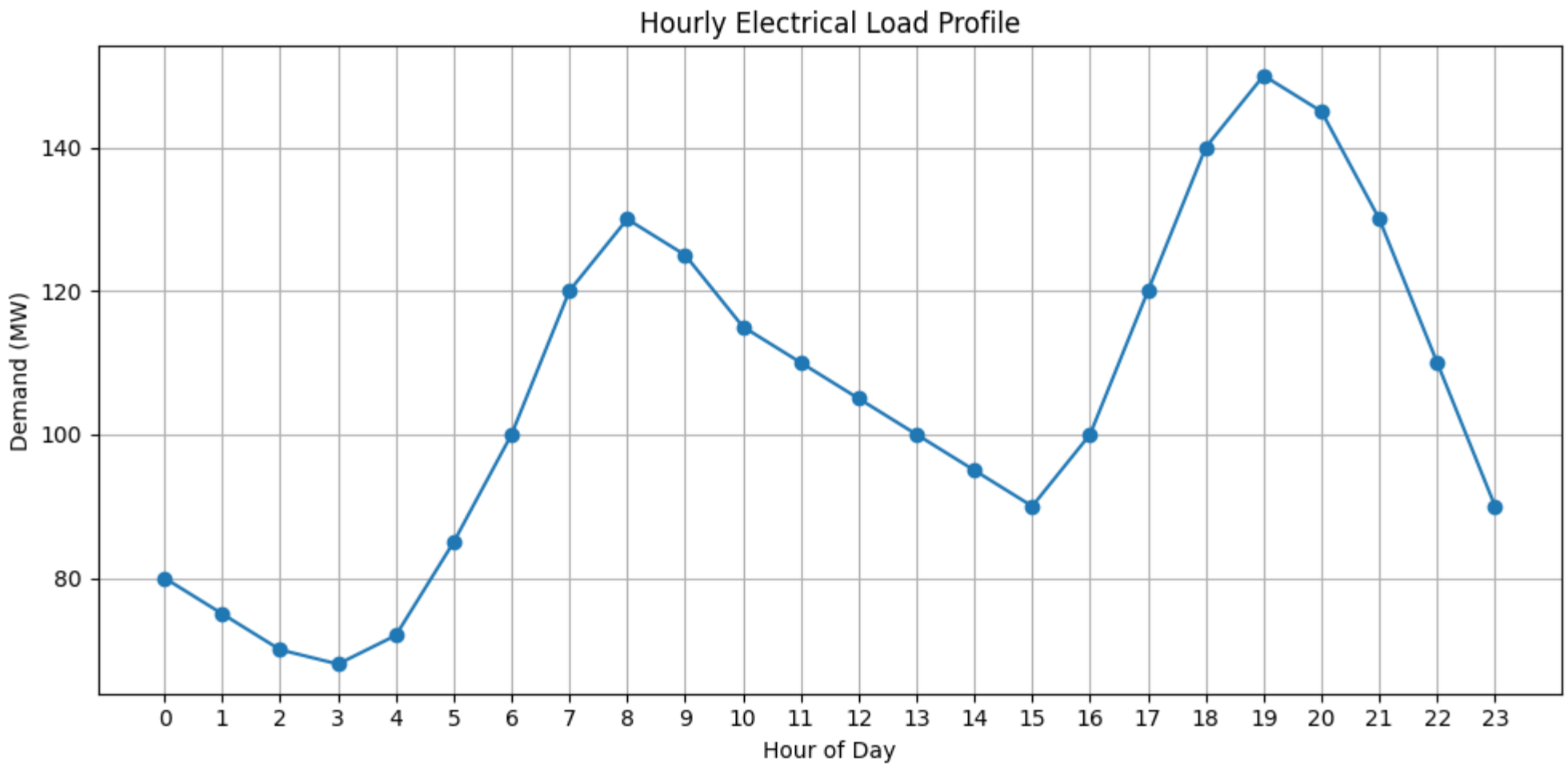
1. Hourly demand values (list or array of 24 numbers).

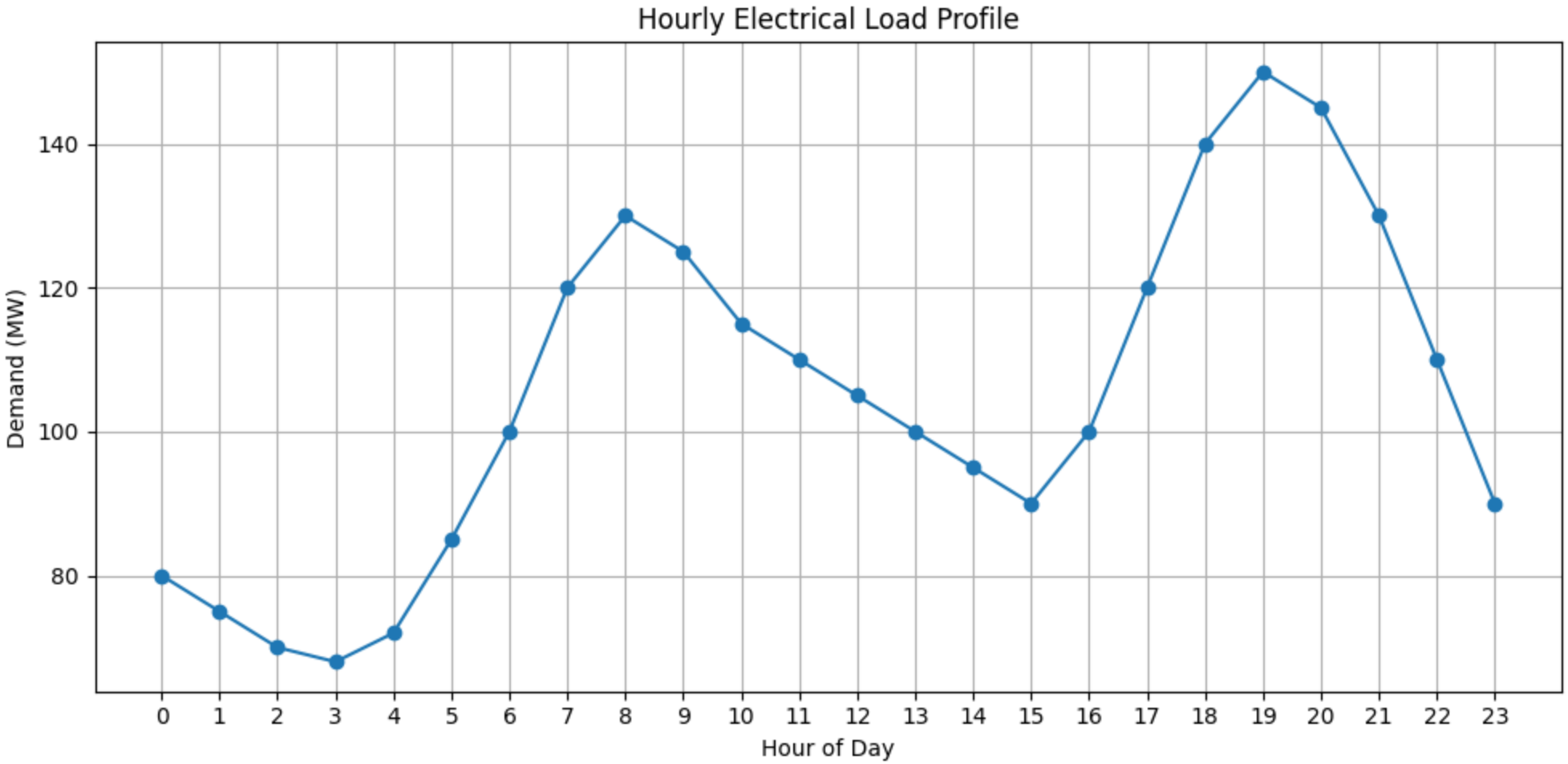
2.Plot using Matplotlib:

```python
import matplotlib.pyplot as plt
import numpy as np

# Example hourly demand (MW)
hours = np.arange(0, 24)
demand = np.array([80, 75, 70, 68, 72, 85, 100, 120,
                   130, 125, 115, 110, 105, 100, 95, 90,
                   100, 120, 140, 150, 145, 130, 110, 90])

plt.figure(figsize=(10, 5))
plt.plot(hours, demand, marker='o', linestyle='-')
plt.title('Hourly Electrical Load Profile')
plt.xlabel('Hour of Day')
plt.ylabel('Demand (MW)')
plt.grid(True)
plt.xticks(hours)
plt.tight_layout()
plt.show()
```


Hourly Electrical Load Profile
Demand (MW)
140
120
100
80
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
Hour of Day


Hourly Electrical Load Profile
Demand (MW)
140
120
100
80
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
Hour of Day