

Medical Statistics and Data Analysis

Lecture 3: Introduction to Python - Programming Fundamentals and Visualization

Zhiyu Wan, PhD (wanzhy@shanghaitech.edu.cn)

Assistant Professor of Biomedical Engineering

ShanghaiTech University

September 28, 2026

Why Python for Medical Data Analysis

- Readable syntax supports students with limited programming experience
- NumPy and pandas provide a complete workflow for numerical and tabular data
- Matplotlib and Seaborn produce reproducible scientific figures
- Jupyter notebooks combine code, output, figures, and interpretation
- The same ecosystem extends naturally to statistics and machine learning

Textbook and Software

Reference

- Python医学数据分析入门
- 赵军、刘文婷编著
- People's Posts and Telecommunications Press, 2022

Course environment

- Python 3
- Jupyter Notebook or JupyterLab
- NumPy, pandas, Matplotlib, Seaborn

Learning Goals of This Lecture

- Read and write basic Python expressions, objects, and control flow
- Use lists, dictionaries, functions, and files
- Manipulate arrays with NumPy and tables with pandas
- Create scientific figures with Matplotlib and Seaborn
- Inspect a medical dataset and summarize numeric variables
- Run and modify a reproducible Jupyter notebook

Course Roadmap

- Python environment, syntax, and core objects
- Data structures, control flow, functions, and files
- NumPy arrays and vectorized numerical operations
- pandas DataFrames, cleaning, reshaping, and file interfaces
- Matplotlib and Seaborn visualization
- Dataset inspection and numerical descriptive statistics
- Integrated practice in a Jupyter notebook

Part I: Python Foundations

Environment, syntax, objects, operators, and help

Python and Jupyter

Python

- Programming language and execution engine
- Objects, functions, modules, and packages
- Used for data analysis and automation

Jupyter

- Interactive notebook environment
- Combines Markdown, code, output, and figures
- Run cells in a visible sequence

The Scientific Python Ecosystem

- NumPy: multidimensional arrays and numerical operations
- pandas: labeled tables and data preparation
- Matplotlib: low-level plotting foundation
- Seaborn: statistical graphics with concise syntax
- SciPy and statsmodels: statistical methods beyond today's scope
- scikit-learn: machine learning beyond Chapter 7.2

Your First Python Commands

```
# A comment starts with #  
2 + 3  
  
message = "Hello, Python!"  
print(message)  
  
import math  
math.sqrt(25)  
help(len)
```

Objects, Names, and Types

```
age = 21  
weight_kg = 63.5  
smoker = False  
patient_id = "P001"
```

```
type(age)  
type(weight_kg)  
type(smoker)  
type(patient_id)
```

Core Data Types

Numbers and logic

- int: 21
- float: 63.5
- bool: True or False
- complex: $2 + 3j$

Text and missingness

- str: 'Control'
- None: absence of a Python object
- numpy.nan: missing numeric value
- Use `type()` and `isinstance()` to inspect

Operators and Comparisons

```
x = 8
```

```
y = 3
```

```
x + y
```

```
x / y
```

```
x // y
```

```
x ** 2
```

```
x >= 5
```

```
(x >= 5) and (y < 5)
```

Strings and Formatting

```
patient = "P001"  
sbp = 128  
  
patient.lower()  
patient.startswith("P")  
  
label = f"{patient}: SBP = {sbp} mmHg"  
print(label)
```

Part II: Data Structures and Control Flow

Lists, tuples, dictionaries, sets, decisions, loops, functions, and files

Lists

```
bp = [118, 125, 132, 121, 140]

len(bp)
bp[0]
bp[-1]
bp[1:4]
bp.append(136)
max(bp)
```

Tuples, Dictionaries, and Sets

Tuple and dictionary

- Tuple: ordered and usually unchanged
- Dictionary: key-value pairs
- Useful for coordinates, settings, and patient records

Set

- Stores unique elements
- Supports union and intersection
- Useful for IDs, categories, and duplicate checks

Indexing and Slicing

```
values = [10, 20, 30, 40, 50]
```

```
values[0]
```

```
values[-1]
```

```
values[1:4]
```

```
values[:3]
```

```
values[::-2]
```

```
record = {"id": "P01", "age": 67}
```

```
record["age"]
```

Conditional Statements

```
risk_score = 8

if risk_score >= 10:
    category = "High"
elif risk_score >= 5:
    category = "Moderate"
else:
    category = "Low"

category
```

Loops

```
values = [4, 7, 10]
result = []

for value in values:
    result.append(value ** 2)

for index, value in enumerate(values):
    print(index, value)
```

Comprehensions

```
values = [4, 7, 10]

squares = [x ** 2 for x in values]
large = [x for x in values if x >= 7]

labels = {x: "high" if x >= 7 else "low"
          for x in values}
```

Writing a Function

```
def summarize_bp(values):  
    clean = [x for x in values if x is not None]  
    return {  
        "n": len(clean),  
        "mean": sum(clean) / len(clean)  
    }  
  
summarize_bp([118, 125, None, 132])
```

Function Arguments

Common forms

- Positional and named arguments
- Default parameter values
- *args for extra positional arguments
- **kwargs for extra named arguments

Good practice

- Focused responsibility
- Meaningful names and docstrings
- Predictable return type
- Avoid hidden dependence on global variables

Lambda Functions

```
measurements = [("P01", 128),  
                ("P02", 142),  
                ("P03", 118)]  
  
ordered = sorted(  
    measurements,  
    key=lambda item: item[1]  
)  
ordered
```

Text and CSV Files

```
from pathlib import Path
import csv

Path("output").mkdir(exist_ok=True)

with open("output/notes.txt", "w",
          encoding="utf-8") as file:
    file.write("Reproducible analysis\n")
```

Part III: NumPy Foundations

Arrays, indexing, reshaping, vectorization, statistics, and matrix operations

Creating NumPy Arrays

```
import numpy as np

a = np.array([1, 2, 3, 4])
zeros = np.zeros((2, 3))
ones = np.ones((2, 3))
sequence = np.arange(0, 10, 2)
grid = np.linspace(0, 1, 6)
```

Shape, Size, Dimensions, and dtype

```
matrix = np.arange(12).reshape(3, 4)

matrix.shape
matrix.size
matrix.ndim
matrix.dtype

matrix.astype(float)
```

Array Indexing and Slicing

```
matrix = np.arange(12).reshape(3, 4)

matrix[0, 0]
matrix[1, :]
matrix[:, 2]
matrix[0:2, 1:3]
matrix[matrix >= 6]
```

Reshape, Transpose, and Axes

```
a = np.arange(12)
matrix = a.reshape(3, 4)

transposed = matrix.T
flattened = matrix.ravel()

matrix.sum(axis=0) # by column
matrix.sum(axis=1) # by row
```

Vectorized Operations

```
x = np.array([4, 7, 10])  
  
x + 2  
x ** 2  
np.sqrt(x)  
x >= 7  
np.where(x >= 7, "high", "low")
```

NumPy Descriptive Statistics

```
x = np.array([118, 125, 132, 121, 140])  
  
np.mean(x)  
np.median(x)  
np.std(x, ddof=1)  
np.quantile(x, [0.25, 0.75])  
np.min(x), np.max(x)
```

Matrix Operations

```
A = np.array([[1, 2], [3, 4]])  
B = np.array([[2, 0], [1, 2]])  
  
A * B          # element-wise  
A @ B          # matrix product  
A.T            # transpose  
np.linalg.det(A)  
np.linalg.inv(A)
```

Reproducible Random Numbers

```
rng = np.random.default_rng(110)

normal = rng.normal(0, 1, size=100)
integers = rng.integers(0, 10, size=8)
selected = rng.choice(["A", "B", "C"], size=5)
```

Saving and Loading Arrays

```
array = np.arange(12).reshape(3, 4)

np.save("output/array.npy", array)
restored = np.load("output/array.npy")

np.savetxt("output/array.csv", array,
           delimiter=",")
```

Part IV: pandas for Medical Tables

Series, DataFrames, selection, cleaning, merging, reshaping, and file interfaces

Series and DataFrame

Series

- One-dimensional labeled values
- Has an index and a dtype
- Often represents one variable

DataFrame

- Two-dimensional labeled table
- Columns may have different dtypes
- Usually represents a complete dataset

A Small Medical DataFrame

```
import pandas as pd

patients = pd.DataFrame({
    "id": ["P01", "P02", "P03", "P04"],
    "age": [45, 67, 52, 71],
    "group": ["Control", "Treatment",
              "Treatment", "Control"],
    "sbp": [118, 142, 135, 128]
})
patients
```

Inspecting a DataFrame

```
patients.head()  
patients.tail()  
patients.shape  
patients.columns  
patients.dtypes  
patients.info()  
patients.describe()
```

Selecting Columns and Rows

```
patients["sbp"]  
patients[["age", "sbp"]]  
  
patients.loc[patients["age"] >= 65,  
             ["id", "age", "sbp"]]  
  
patients.iloc[0:3, 0:2]
```

Filtering with Conditions

```
older = patients.loc[patients["age"] >= 65]

treated = patients.query(
    "group == 'Treatment' and sbp >= 130"
)

selected = patients.loc[
    patients["group"].isin(["Control"])
]
```

Assign, Rename, and Sort

```
patients = patients.assign(  
    pulse_pressure=patients["sbp"] - 80  
)  
  
patients = patients.rename(  
    columns={"sbp": "sbp_mmHg"}  
)  
  
patients = patients.sort_values(  
    ["group", "age"]  
)
```

Identifying Missing Values

```
patients.isna()
patients.isna().sum()
patients.notna().sum()

missing_rows = patients.loc[
    patients["sbp_mmHg"].isna()
]
```

Handling Missing Values

```
complete = patients.dropna(  
    subset=["sbp_mmHg"]  
)  
  
median_sbp = patients["sbp_mmHg"].median()  
patients["sbp_filled"] = (  
    patients["sbp_mmHg"].fillna(median_sbp)  
)
```

Duplicates and Data Types

```
patients.duplicated().sum()
patients = patients.drop_duplicates()

patients["group"] = patients["group"].astype("category")
patients["age"] = pd.to_numeric(
    patients["age"], errors="coerce"
)
```

Merging and Concatenating

```
visits = pd.DataFrame({
    "id": ["P01", "P02", "P03"],
    "visit": [1, 1, 1]
})

merged = patients.merge(
    visits, on="id", how="left", validate="one_to_one"
)

combined = pd.concat([patients, patients],
                      ignore_index=True)
```

Wide and Long Formats

```
long = pd.melt(
    patients,
    id_vars=["id", "group"],
    value_vars=["age", "sbp_mmHg"],
    var_name="measure",
    value_name="value"
)

wide = long.pivot(
    index="id", columns="measure", values="value"
)
```

Importing and Exporting Data

```
patients = pd.read_csv(  
    "data/patients.csv",  
    na_values=["NA", "Missing", ""]  
)  
  
patients.to_csv(  
    "output/patients_clean.csv",  
    index=False  
)  
  
# pd.read_excel("data/patients.xlsx")
```

Part V: Scientific Data Visualization

Matplotlib foundations and statistical graphics with Seaborn

A Scientific Figure Has a Job

- Show a distribution, comparison, relationship, composition, or change over time
- Choose a graphic that matches variable types
- Label axes with units and define abbreviations
- Show individual observations and uncertainty when useful
- Use color to encode information consistently
- Save figures at the intended publication or presentation size

Matplotlib Figure Anatomy

Figure and axes

- Figure: complete canvas
- Axes: plotting region
- One figure may contain several axes

Graphical elements

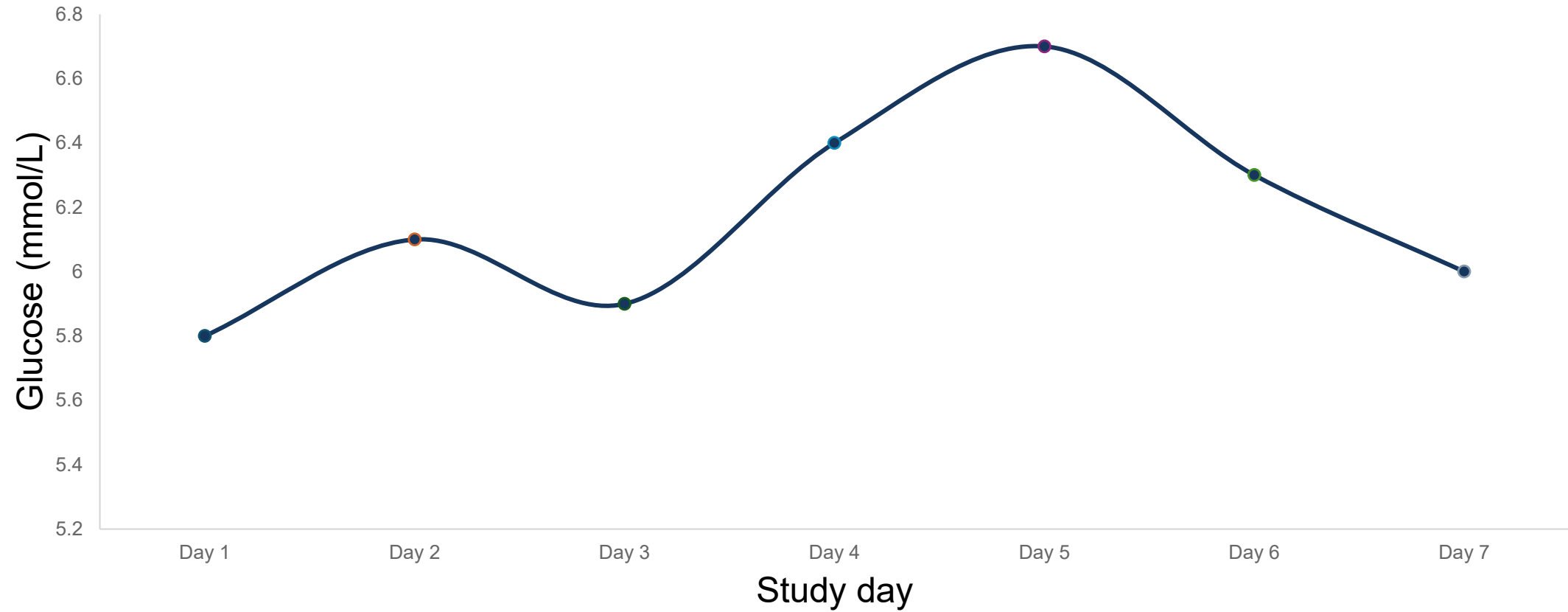
- Lines, points, bars, and patches
- Titles, labels, ticks, and legends
- Styles and layout control presentation

A Reusable Matplotlib Pattern

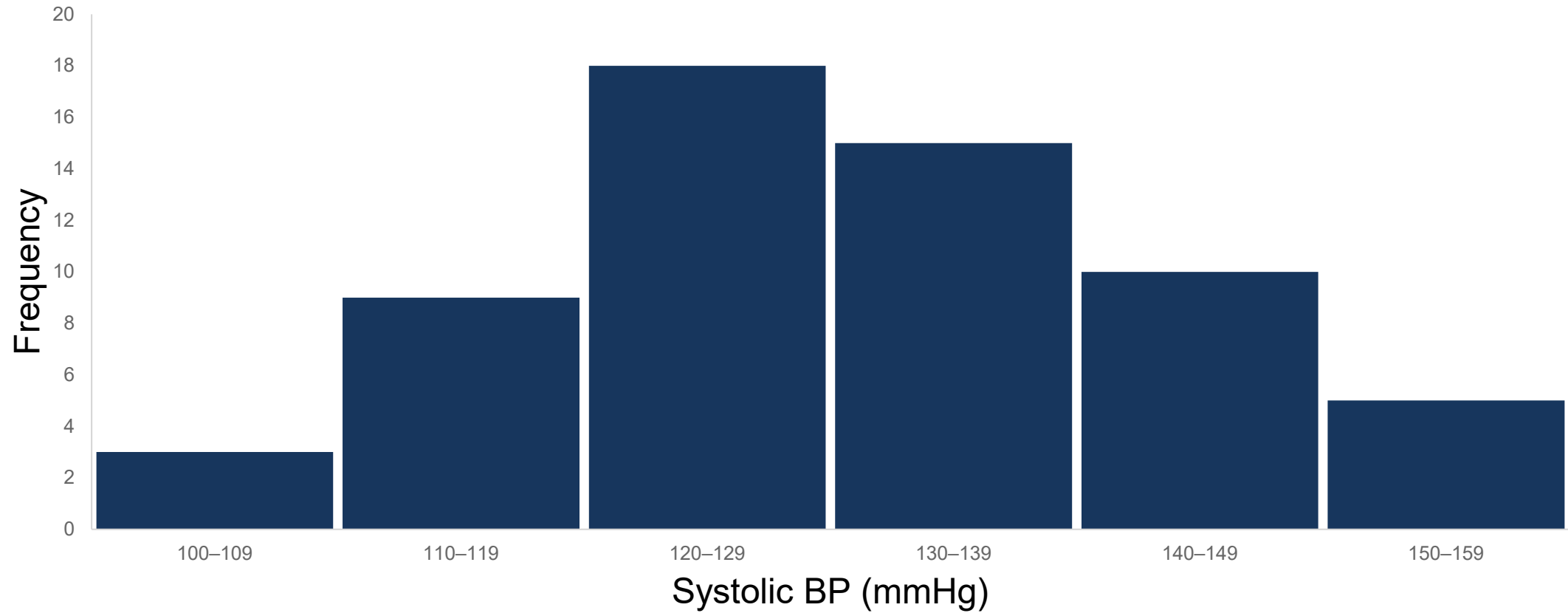
```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(7, 5))
ax.scatter(data["age"], data["sbp"],
           alpha=0.7)
ax.set(
    title="Age and systolic blood pressure",
    xlabel="Age (years)",
    ylabel="Systolic BP (mmHg)"
)
fig.tight_layout()
```

Line Plot



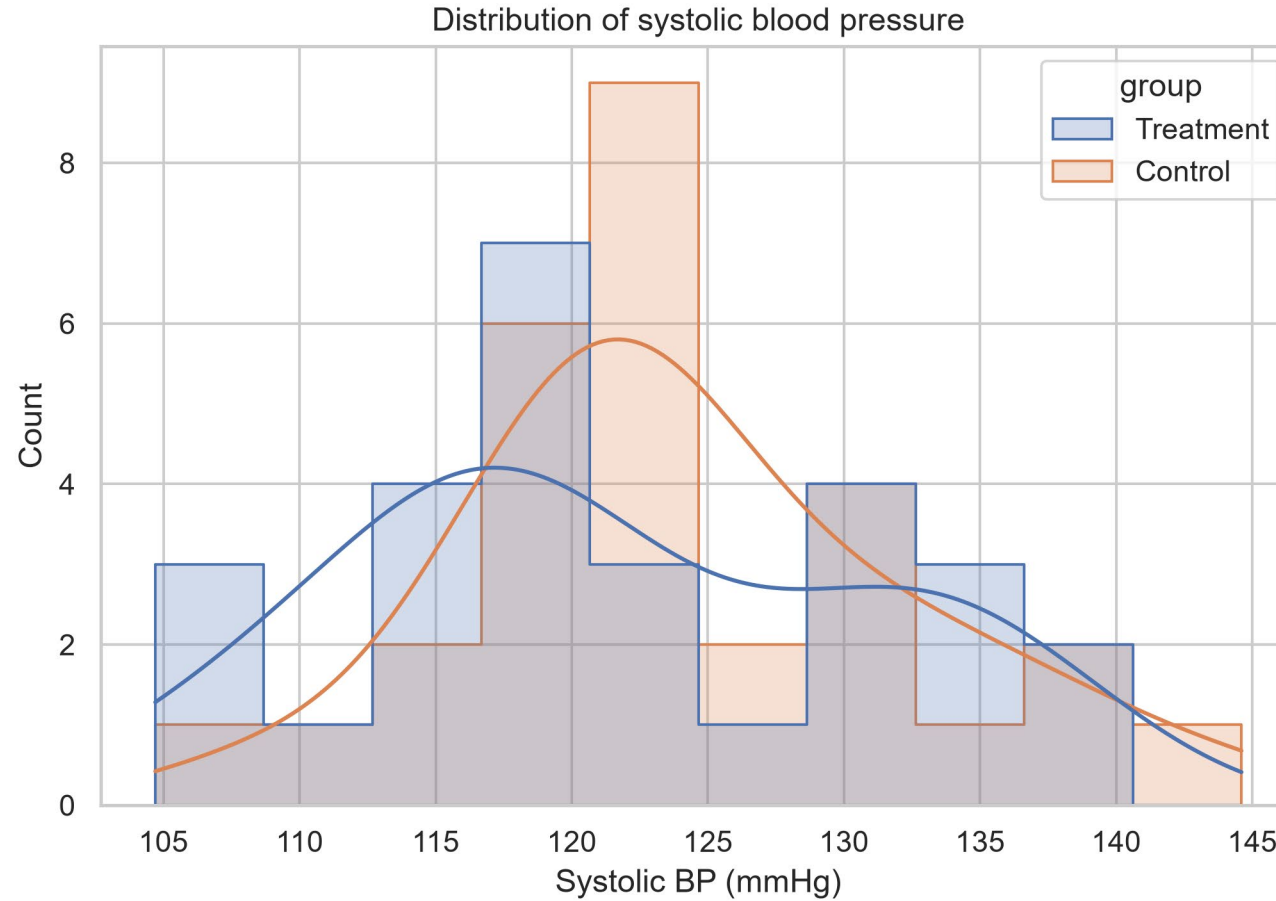
Histogram



Histogram and Density Curve

```
import seaborn as sns
sns.histplot(
    data=patients, x="sbp", hue="group",
    bins=10, kde=True,
    element="step", common_norm=False
)
plt.xlabel("Systolic BP (mmHg)")
plt.ylabel("Count")
```

Histogram and Density Curve



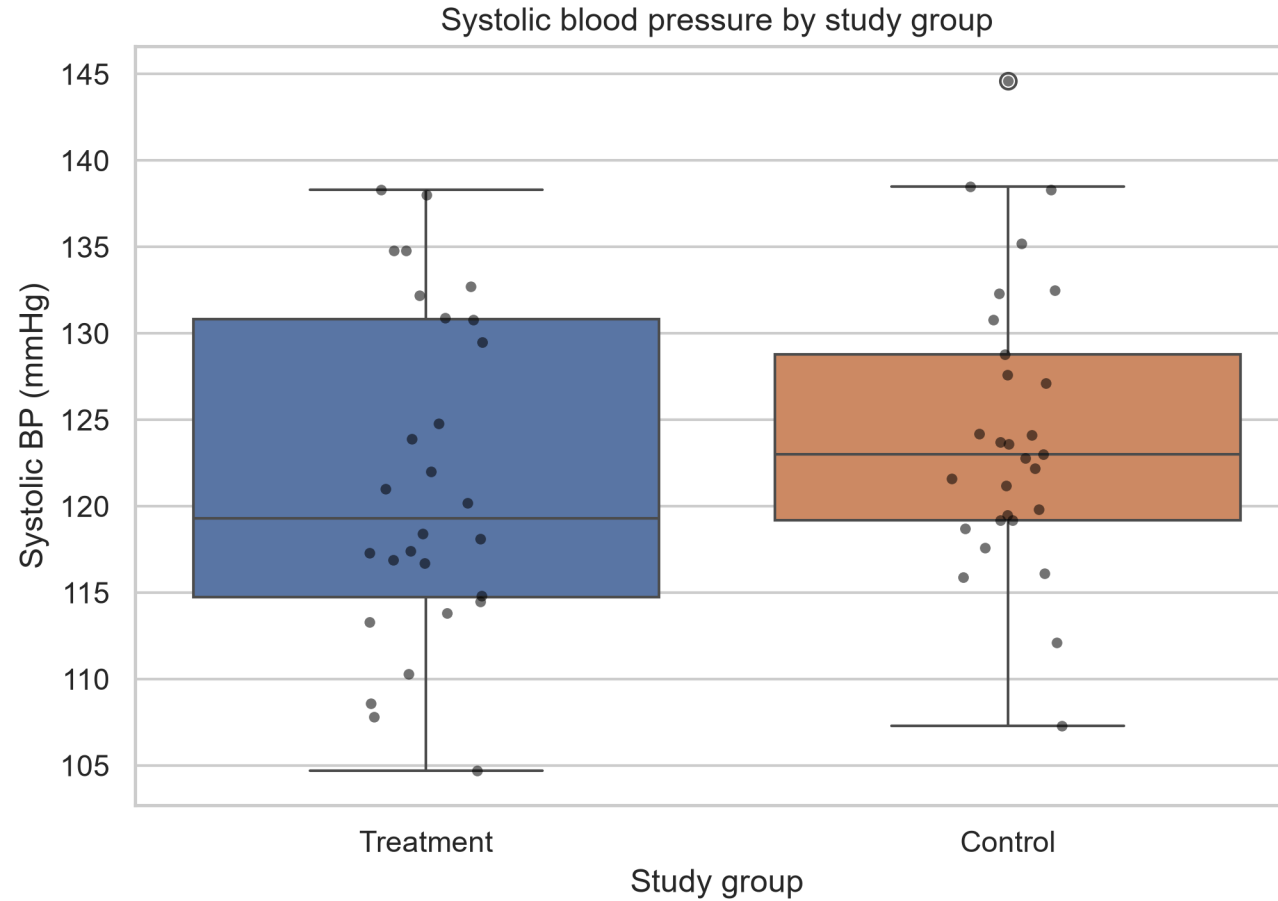
Boxplot and Violin Plot

```
fig, axes = plt.subplots(1, 2, figsize=(10, 4))  
  
sns.boxplot(data=patients, x="group", y="sbp",  
            ax=axes[0])  
sns.violinplot(data=patients, x="group", y="sbp",  
               inner="quart", ax=axes[1])  
  
fig.tight_layout()
```

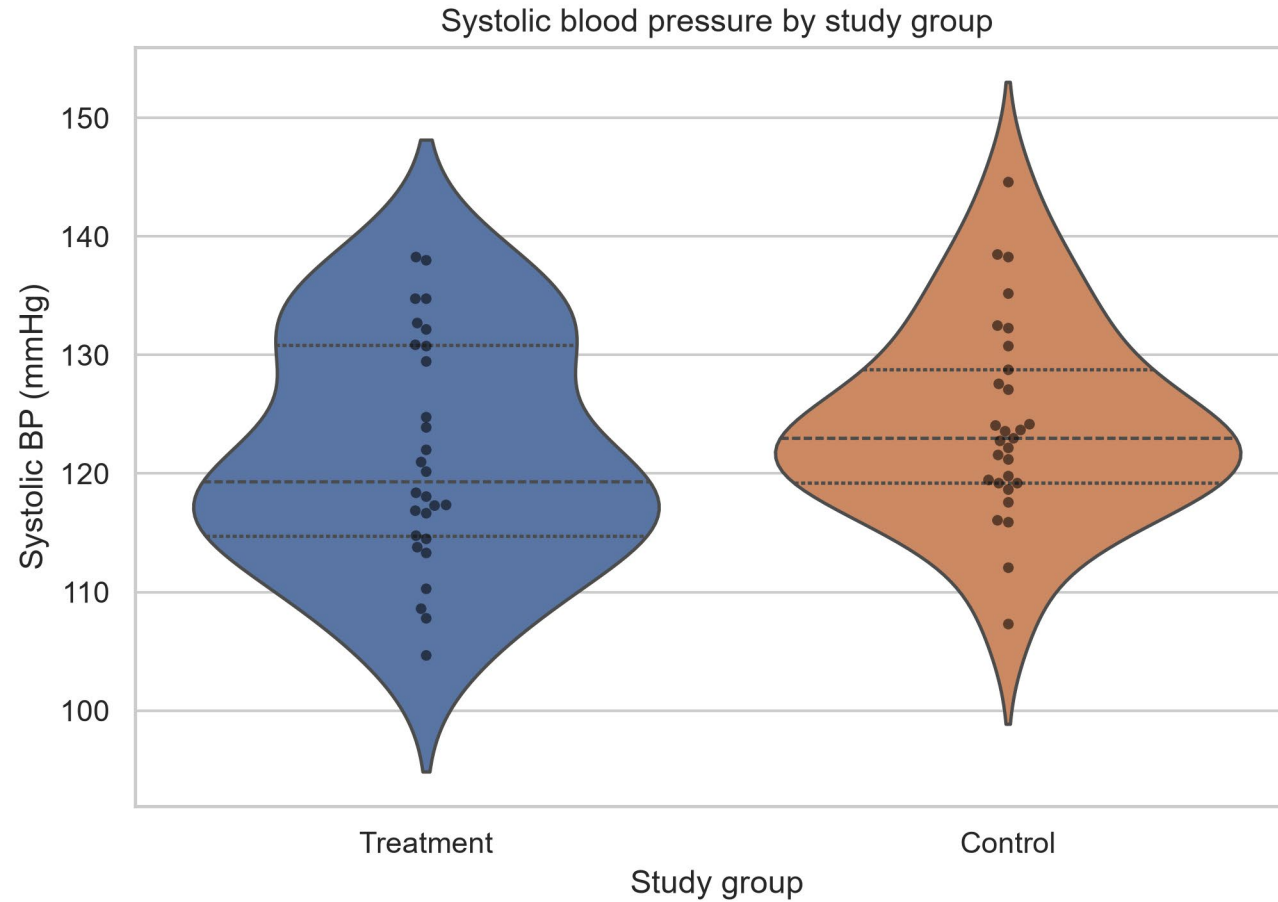
Strip Plot and Swarm Plot

```
fig, axes = plt.subplots(1, 2, figsize=(10, 4))  
  
sns.stripplot(data=patients, x="group", y="sbp",  
              jitter=True, alpha=0.7, ax=axes[0])  
sns.swarmplot(data=patients, x="group", y="sbp",  
              size=4, ax=axes[1])
```

Boxplot and Strip Plot



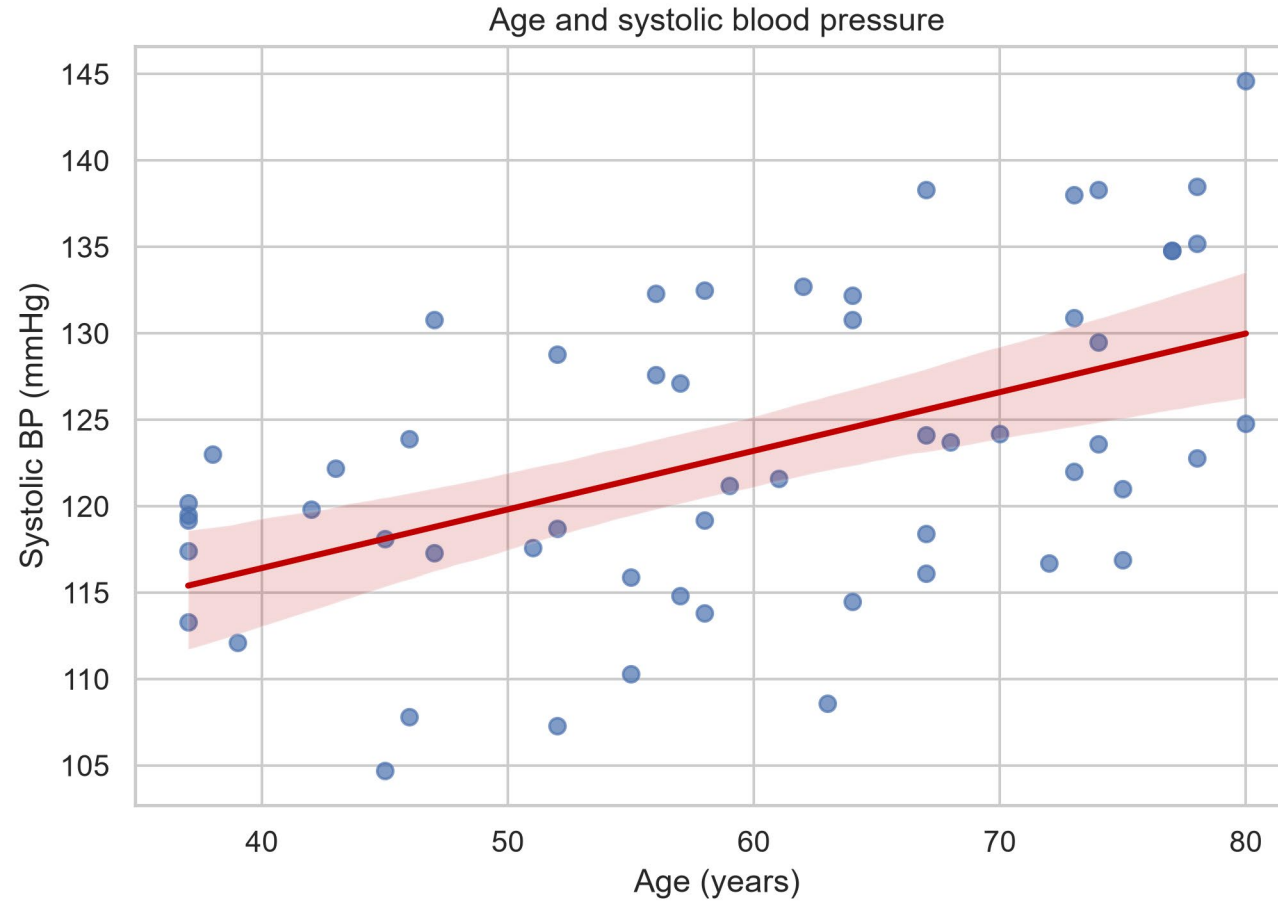
Violin Plot and Swarm Plot



Regression Plot

```
sns.regplot(  
    data=patients, x="age", y="sbp",  
    scatter_kws={"alpha": 0.7},  
    line_kws={"color": "#C00000"}  
)  
plt.xlabel("Age (years)")  
plt.ylabel("Systolic BP (mmHg)")
```

Regression Plot

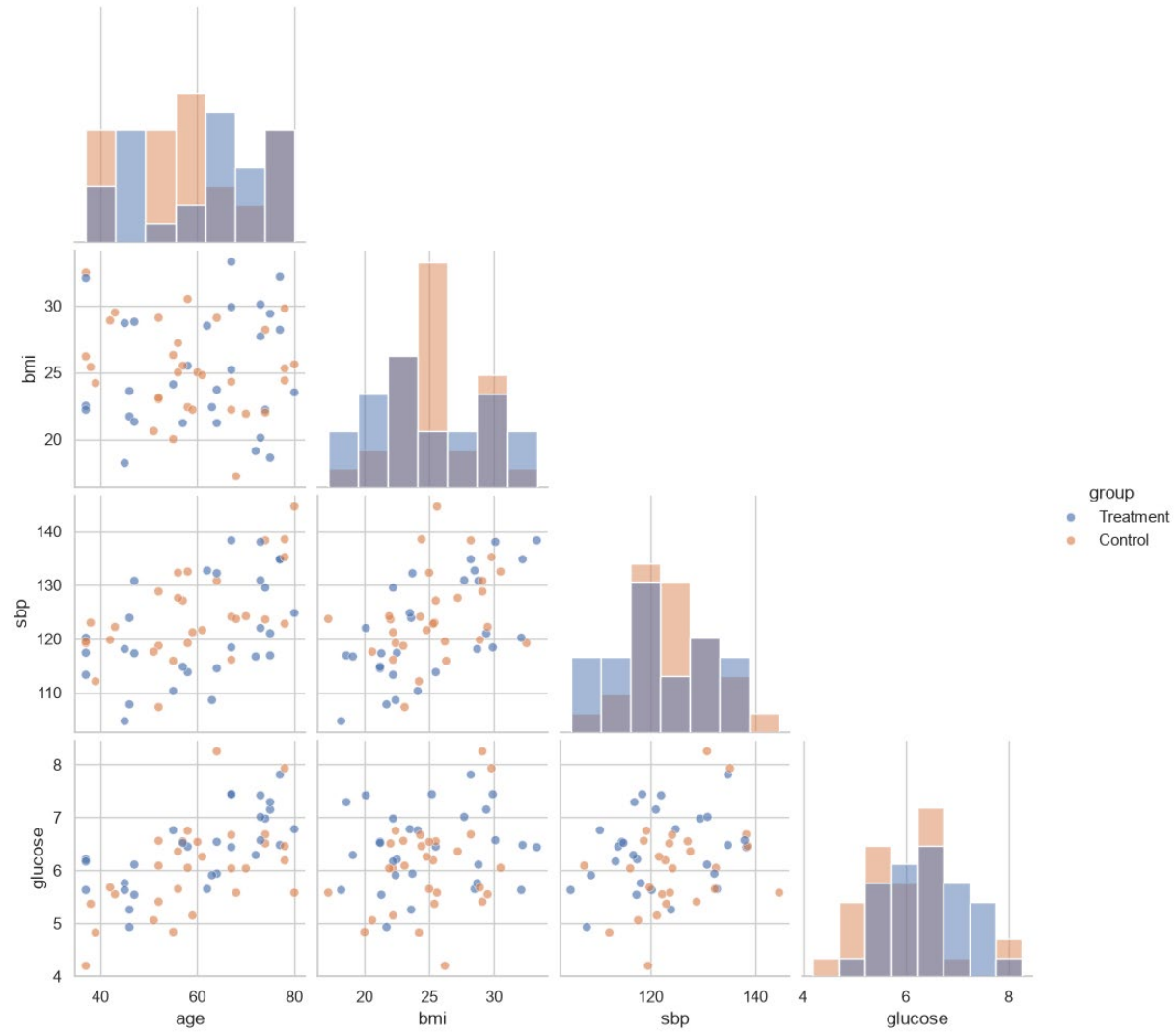


Scatterplot Matrix

```
variables = ["age", "bmi", "sbp", "glucose"]

sns.pairplot(
    patients, vars=variables, hue="group",
    corner=True, diag_kind="hist",
    plot_kws={"alpha": 0.65}
)
```

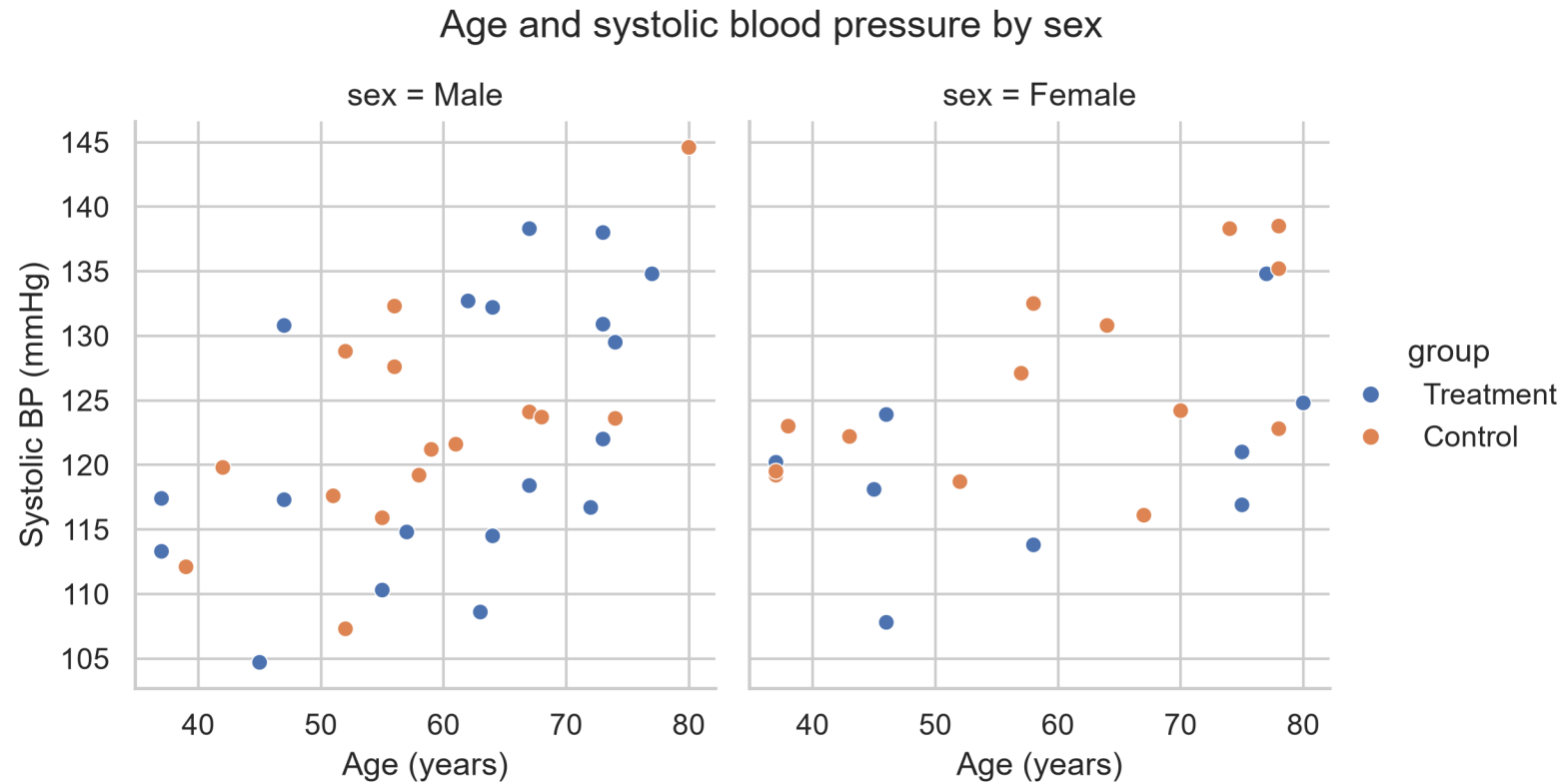
Scatterplot Matrix



Faceting

```
g = sns.relplot(  
    data=patients,  
    x="age", y="sbp",  
    col="sex", hue="group",  
    kind="scatter",  
    height=4, aspect=0.9  
)  
g.set_axis_labels("Age (years)",  
                  "Systolic BP (mmHg)")
```

Faceting

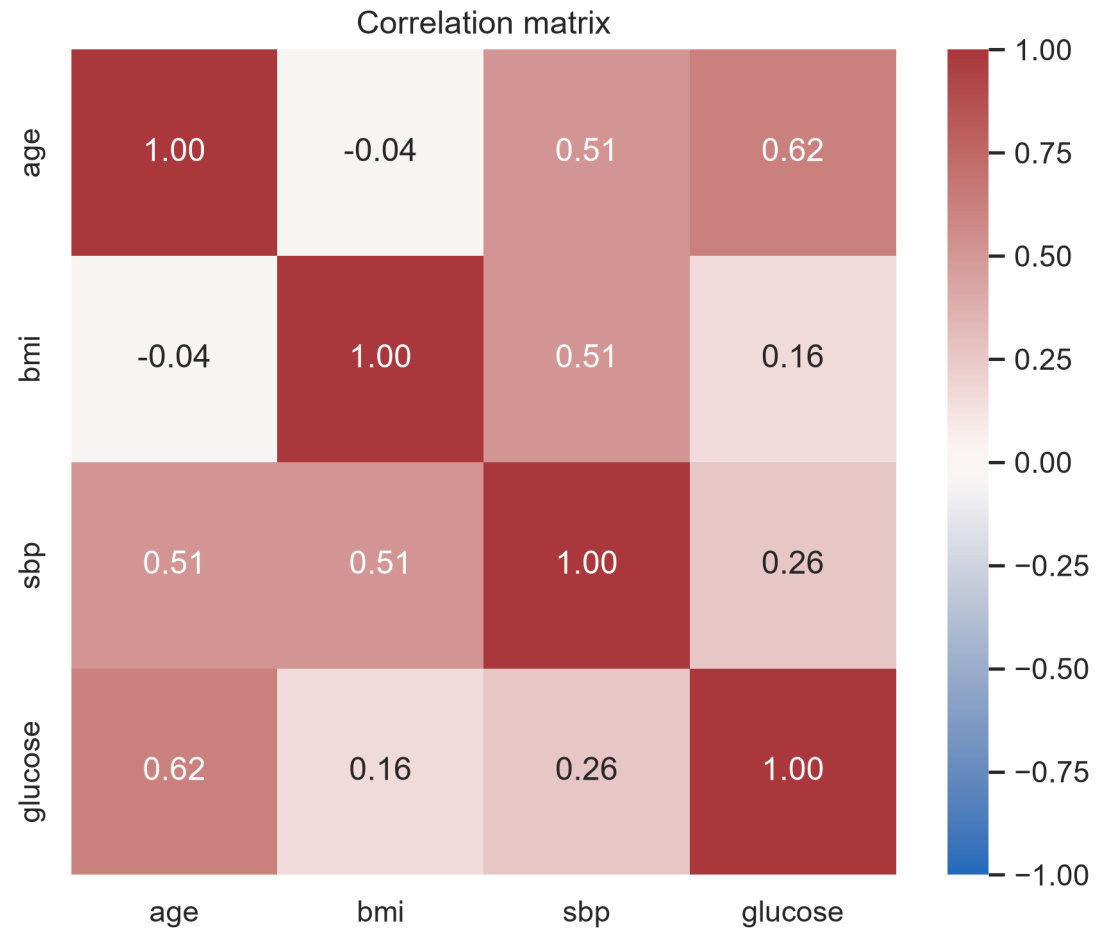


Correlation Heatmap

```
numeric = patients[[
    "age", "bmi", "sbp", "glucose"
]]
corr = numeric.corr()

sns.heatmap(
    corr, annot=True, fmt=".2f",
    cmap="vlag", center=0,
    vmin=-1, vmax=1, square=True
)
```

Correlation Heatmap

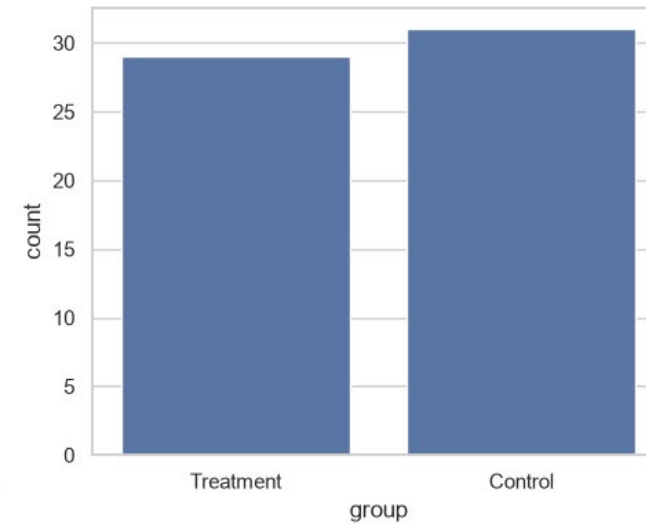
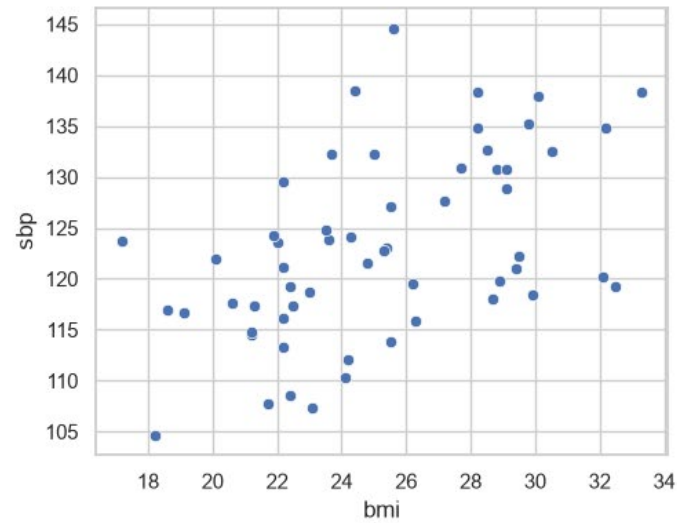
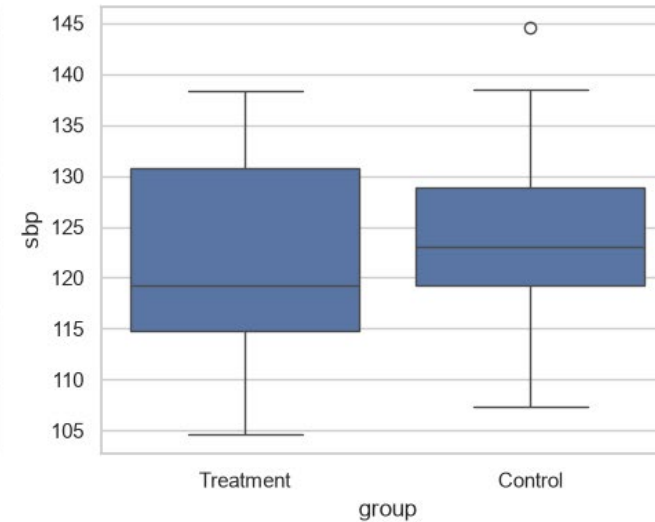
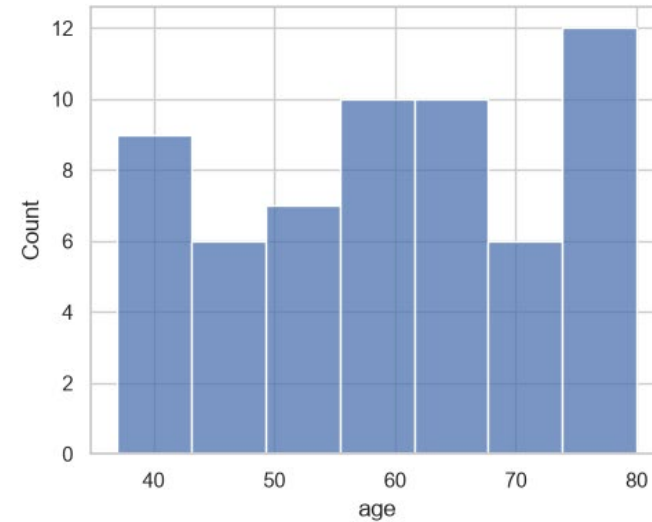


Multiple Panels

```
fig, axes = plt.subplots(2, 2, figsize=(10, 8))

sns.histplot(data=patients, x="age", ax=axes[0, 0])
sns.boxplot(data=patients, x="group", y="sbp",
            ax=axes[0, 1])
sns.scatterplot(data=patients, x="bmi", y="sbp",
               ax=axes[1, 0])
sns.countplot(data=patients, x="group",
              ax=axes[1, 1])
fig.tight_layout()
```

Multiple Panels



Saving Figures

```
from pathlib import Path
Path("figures").mkdir(exist_ok=True)

fig.savefig(
    "figures/patient_summary.png",
    dpi=300, bbox_inches="tight",
    facecolor="white"
)

# Vector format for editable publication graphics
fig.savefig("figures/patient_summary.svg",
            bbox_inches="tight")
```

Matplotlib and Seaborn

Matplotlib

- Precise control over every graphical element
- Best foundation for custom layouts
- Object-oriented interface scales to complex figures

Seaborn

- Concise statistical graphics
- Integrates directly with pandas DataFrames
- Built-in grouping, estimation, and faceting

Part VI: Dataset Inspection and Descriptive Statistics

Data quality checks and numerical summaries

Dataset Information

```
patients.shape  
patients.info()  
patients.head()  
patients.sample(5, random_state=110)  
patients.nunique()  
patients.isna().sum()  
patients.describe(include="all")
```

Numerical Descriptive Statistics

```
patients["sbp"].count()  
patients["sbp"].mean()  
patients["sbp"].median()  
patients["sbp"].std()  
patients["sbp"].min()  
patients["sbp"].max()  
patients["sbp"].quantile([0.25, 0.75])
```

Grouped Descriptive Statistics

```
summary = (  
    patients  
    .groupby("group", observed=True)["sbp"]  
    .agg(["count", "mean", "median", "std",  
         "min", "max"])  
    .round(2)  
)  
summary
```

Quantiles and the IQR Rule

```
q1 = patients["sbp"].quantile(0.25)
q3 = patients["sbp"].quantile(0.75)
iqr = q3 - q1

lower = q1 - 1.5 * iqr
upper = q3 + 1.5 * iqr

outliers = patients.loc[
    ~patients["sbp"].between(lower, upper)
]
```

Skewness and Distribution Shape

```
patients[["age", "bmi", "sbp", "glucose"]].agg(  
    ["mean", "median", "std", "skew"]  
).T  
  
sns.histplot(data=patients, x="glucose", kde=True)
```

Part VII: Integrated Practice

A small medical dataset from generation to interpretation

The Practice Dataset

Study context

- 60 simulated adult participants
- Control and Treatment groups
- Male and female participants
- Fixed random seed for reproducibility

Variables

- age, sex, group
- BMI, systolic blood pressure
- Glucose and heart rate
- A small number of deliberate missing values

An Integrated pandas Pipeline

```
summary = (  
    patients  
    .dropna(subset=["sbp"])  
    .assign(bmi_group=lambda d: pd.cut(  
        d["bmi"], bins=[0, 25, 30, np.inf],  
        labels=["Normal", "Overweight", "Obesity"]  
    ))  
    .groupby(["group", "bmi_group"], observed=True)  
    .agg(n=("sbp", "size"),  
        mean_sbp=("sbp", "mean"),  
        sd_sbp=("sbp", "std"))  
    .reset_index()  
)
```

Integrated Practice

- Run all notebook cells from top to bottom
- Inspect dimensions, data types, and missing values
- Calculate overall and grouped descriptive statistics
- Create a histogram, boxplot, scatterplot, and correlation heatmap
- Save at least one figure at 300 dpi
- Write two descriptive sentences without making unsupported causal claims
- Change one plotting choice and explain its effect

Practice Answer Pattern

```
clean = patients.dropna(subset=["sbp"])

summary = clean.groupby("group")["sbp"].agg(
    n="count", mean="mean", median="median",
    sd="std", minimum="min", maximum="max"
).round(2)

sns.boxplot(data=clean, x="group", y="sbp")
sns.stripplot(data=clean, x="group", y="sbp",
              color="black", alpha=0.55)
```

Common Errors and Quick Checks

- `NameError`: check spelling, capitalization, and execution order
- `IndentationError`: align code blocks consistently
- `KeyError`: inspect DataFrame column names
- `TypeError`: inspect object types before operating
- `ValueError`: inspect shapes, lengths, and invalid values
- Missing results: check NaN values and aggregation options
- Unexpected plot: inspect mappings, grouping, and dtypes

A Reproducible Notebook

```
# 1. State the purpose and question  
# 2. Import packages  
# 3. Set the random seed  
# 4. Load or generate data  
# 5. Inspect and clean  
# 6. Transform and summarize  
# 7. Visualize and save figures  
# 8. Interpret results
```

```
import sys  
print(sys.version)  
print(pd.__version__)
```

Key Takeaways

- Python objects and data structures determine how code behaves
- NumPy arrays support efficient numerical operations
- pandas provides a transparent workflow for medical tables
- Matplotlib and Seaborn connect data structures with scientific figures
- Dataset inspection must precede statistical calculation
- Descriptive statistics should report center, spread, range, and sample size
- The notebook provides an executable bridge from code to interpretation

Questions

Run the notebook, modify one analysis choice, and explain what changes