**Q1: Statistics Summary and Real-World Use Cases**
**What is Statistics?**
Statistics is the science of collecting, analyzing, interpreting, and presenting data to make informed decisions and understand patterns in complex information.
**5 Real-World Use Cases of Statistics in Data Science:**

1. **Netflix Recommendation System**: Uses statistical algorithms to analyze viewing patterns and recommend content based on user behavior and preferences.
2. **Medical Drug Testing**: Pharmaceutical companies use statistical methods to test drug efficacy, determine proper dosages, and identify side effects through clinical trials.
3. **Credit Risk Assessment**: Banks use statistical models to evaluate loan applications, predict default probabilities, and set interest rates based on customer financial history.
4. **Quality Control in Manufacturing**: Companies like Toyota use statistical process control to monitor production quality, detect defects early, and maintain consistent product standards.
5. **Sports Analytics**: Teams like the Golden State Warriors use statistical analysis to evaluate player performance, develop game strategies, and make data-driven decisions about trades and drafts.

**Q2: Data Types and Applications**
**Structured vs Unstructured Data**
**Structured Data:**

- Organized in a predefined format (tables, databases)
- Easy to search, analyze, and process
- Examples: Excel spreadsheets, SQL databases, CSV files
- Format: Rows and columns with clear relationships

**Unstructured Data:**

- No predefined format or organization
- Requires preprocessing before analysis
- Examples: Social media posts, emails, images, videos, audio files
- Format: Text documents, multimedia files, web pages

**Applications of Statistics in Different Fields:**
**Healthcare:**

- Clinical trial analysis to test new treatments
- Epidemiological studies to track disease spread
- Patient outcome prediction and risk assessment

**Business:**

- Sales forecasting and inventory management
- Customer segmentation and market research
- Financial performance analysis and budgeting

**Marketing:**

- A/B testing for campaign effectiveness
- Customer lifetime value calculation
- Social media sentiment analysis

**Employee Sales Data Example (Python):**

```python
# ------------- Tiny static dataset -------------
# A list of dictionaries = small "employee sales" table; STATIC, non-random.
employee_sales = [
    {"Name": "Aisha Khan",  "Age": 29, "Region": "North", "MonthlySales": 38000},
    {"Name": "Ben Carter",  "Age": 34, "Region": "South", "MonthlySales": 42500},
    {"Name": "Chao Liu",    "Age": 26, "Region": "East",  "MonthlySales": 33200},
    {"Name": "Diana Lopez", "Age": 41, "Region": "West",  "MonthlySales": 57000},
    {"Name": "Evan Singh",  "Age": 31, "Region": "North", "MonthlySales": 45800},
    {"Name": "Farah Noor",  "Age": 28, "Region": "South", "MonthlySales": 39900},
]

# ------------- Display -------------
# We use pandas only to print a nice table.
import pandas as pd

# DataFrame(...) converts the list of dicts into a table
df = pd.DataFrame(employee_sales)
print(df)
```

### Q.3 — dataframe_dtypes_and_select_dtypes.py

```
# --------------------------------------
# Step 1: Import pandas and create a DataFrame
# --------------------------------------

employees = pd.DataFrame([
    {"Name": "Aisha Khan",   "Age": 29, "Department": "Sales",      "Salary": 72000},
    {"Name": "Ben Carter",   "Age": 34, "Department": "Marketing",  "Salary": 68000},
    {"Name": "Chao Liu",     "Age": 26, "Department": "Sales",      "Salary": 61000},
    {"Name": "Diana Lopez",  "Age": 41, "Department": "Operations", "Salary": 88000},
    {"Name": "Evan Singh",   "Age": 31, "Department": "Sales",      "Salary": 75000},
    {"Name": "Farah Noor",   "Age": 28, "Department": "Marketing",  "Salary": 64000},
])

# Show the first rows to verify the table
print(employees.head())  # .head() shows the top 5 rows by default; no arguments used here


# --------------------------------------
# Step 2: Inspect data types (dtypes)
# --------------------------------------
print(employees.dtypes)


# --------------------------------------
# Step 3: Filter by dtype using select_dtypes
# --------------------------------------
# Select only numeric columns:
numeric_cols = employees.select_dtypes(include=['number'])
#numeric_cols = employees.select_dtypes(include=['int64', 'float64'])
print(numeric_cols)
print(list(numeric_cols.columns))


# Select only categorical (non-numeric) columns:
categorical_cols = employees.select_dtypes(exclude=['number'])
#categorical_cols = employees.select_dtypes(include=['object'])
print(categorical_cols)
print(list(categorical_cols.columns))
```

**Q.4 — Plots: histogram (Age), bar chart (Department count), pie chart (Department %), boxplot (Salary)**

Below, I use **Matplotlib** only (simple and built-in).

If you prefer Seaborn, I add optional short versions after each plot.

```
import pandas as pd                # for table handling
import matplotlib.pyplot as plt    # Matplotlib for plotting

# ------------------ Static dataset for Q4 (separate from other files) ------------------
staff_q4 = [
    {"Name": "Hira Patel",    "Age": 25, "Department": "Sales",      "Salary": 60000},
    {"Name": "Ivan Petrov",   "Age": 29, "Department": "Sales",      "Salary": 65500},
    {"Name": "Julia Stone",   "Age": 31, "Department": "Marketing",  "Salary": 68000},
    {"Name": "Kofi Mensah",   "Age": 38, "Department": "Operations", "Salary": 82000},
    {"Name": "Lina Chen",     "Age": 27, "Department": "Sales",      "Salary": 63000},
    {"Name": "Mohammed Ali",  "Age": 41, "Department": "Finance",    "Salary": 90000},
    {"Name": "Nora Silva",    "Age": 33, "Department": "Marketing",  "Salary": 70000},
    {"Name": "Omar Hassan",   "Age": 36, "Department": "Operations", "Salary": 79000},
]

# Create DataFrame for plotting
df = pd.DataFrame(staff_q4)  # DataFrame(...) constructs a table from our static list

# Optional: set a readable default figure size (applies to all subsequent plots)
plt.rcParams["figure.figsize"] = (6, 4)  # (width, height) in inches


# ------------------ 1) Histogram of Age ------------------
# plt.hist(x, bins=..., edgecolor=...) draws a histogram of numeric values.
#   - x=df["Age"] provides the ages to bin on the x-axis.
#   - bins=5 splits the age range into 5 equal-width buckets.
#   - edgecolor='black' draws borders around bars for readability.

plt.hist(df["Age"], bins=5, edgecolor='black')
plt.title("Age Distribution")           # Title text for the figure
plt.xlabel("Age (years)")               # Label text on the x-axis
plt.ylabel("Number of Employees")       # Label text on the y-axis
plt.tight_layout()                      # Adjust layout so labels/titles fit
plt.show()                              # Render the histogram window


# ------------------ 2) Bar chart: Department counts ------------------
# value_counts() computes category counts; index = names, values = counts.
# counts per department
dept_counts = df["Department"].value_counts()

# plt.bar(x, height) draws bars; here:
#   - x = dept_counts.index (the department names)
#   - height = dept_counts.values (how many employees per department)

plt.bar(dept_counts.index, dept_counts.values)
plt.title("Employees per Department")
plt.xlabel("Department")
plt.ylabel("Count")
plt.xticks(rotation=0)  # keep x labels horizontal
plt.tight_layout()
plt.show()
```

```python
# ------------------ 3) Pie chart: Department share (%) ------------------
# plt.pie(x, labels=..., autopct=...) draws a pie chart.
#   - x = dept_counts.values gives slice sizes.
#   - labels = dept_counts.index assigns labels to slices.
#   - autopct='%1.1f%%' prints percent with 1 decimal place on each slice.

plt.pie(dept_counts.values, labels=dept_counts.index, autopct='%1.1f%%')
plt.title("Department Share (%)")
plt.tight_layout()
plt.show()


# ------------------ 4) Boxplot: Salary distribution ------------------
# plt.boxplot(x) shows median, quartiles, whiskers, and potential outliers.
#   - x = df["Salary"] is the numeric series to summarize.

plt.boxplot(df["Salary"])
plt.title("Salary Distribution")
plt.ylabel("Annual Salary (USD)")
plt.tight_layout()
plt.show()


# ------------------ Quick 1–2 line interpretation (printed) ------------------
print("Observations:")
print("- Ages span mid-20s to early-40s; Sales/Marketing skew younger; Finance has a higher salary.")
print("- Sales is the largest department; Operations/Finance are smaller; salaries vary roughly 60k–90k.")
```

**Q5: Population vs Sample Analysis**

```python
import numpy as np   # numerical ops
import pandas as pd  # optional: for a small summary table

# ------------------ Static population of 1,000 scores ------------------
# Choose 20 realistic exam scores and repeat them 50 times (20*50 = 1000). Deterministic.
base_scores = [72, 85, 91, 67, 88, 94, 76, 83, 95, 69,
               81, 77, 90, 86, 73, 92, 64, 79, 87, 84]

# np.tile(array_like, reps) repeats the sequence.
#   - array_like = base_scores (original list)
#   - reps = 50 (number of repetitions)
population = np.tile(base_scores, 50)  # length will be 1000

print(f"Population size: {len(population)}")  # sanity check -> 1000

# ------------------ Deterministic sample of size 100 ------------------
# Take every 10th element using slicing with a step:
#   - start=0 (begin at first element)
#   - stop omitted (go to end)
#   - step=10 (pick every 10th -> 1000 / 10 = 100 elements)
sample = population[0:None:10]
print(f"Sample size: {len(sample)}")  # sanity check -> 100

# ------------------ Means ------------------
# np.mean(array) computes arithmetic mean (average)
pop_mean = np.mean(population)  # population mean
samp_mean = np.mean(sample)     # sample mean

# ------------------ Standard deviations ------------------
# np.std(array, ddof=...) computes standard deviation.
#   - ddof=0 => population formula (divide by N).
#   - ddof=1 => sample formula (unbiased estimator; divide by N-1).
pop_std = np.std(population, ddof=0)  # population SD
samp_std = np.std(sample, ddof=1)     # sample SD (unbiased)

# ------------------ Print results ------------------
print(f"Population mean: {pop_mean:.2f}")
print(f"Population std : {pop_std:.2f}")
print(f"Sample mean     : {samp_mean:.2f}")
print(f"Sample std      : {samp_std:.2f}")

# ------------------ Optional: tidy table ------------------
summary = pd.DataFrame({
    "Dataset": ["Population", "Sample"],
    "Size": [len(population), len(sample)],
    "Mean": [round(pop_mean, 2), round(samp_mean, 2)],
    "StdDev": [round(pop_std, 2), round(samp_std, 2)]
})
print("\nSummary table:")
print(summary)

# ------------------ Explanation (printed) ------------------
print("\nExplanation:")
print("- Population uses every score (N=1000); population SD uses ddof=0 (divide by N).")
print("- Sample uses a subset (n=100); sample SD uses ddof=1 (divide by n-1) for an unbiased
estimate.")
```