Session 3 Manual

Topic: *Data Analysis & Visualisation*

Focus: Conditional statements, summarisation with `groupby` & pivot tables, and data visualisation with Pandas, Matplotlib, and Seaborn.

1. Conditional Statements in Pandas

What are Conditional Statements?

Conditional statements allow us to filter, transform, or create new columns based on rules/conditions. They are equivalent to *"if-else"* in normal programming but applied to datasets.

For example: "Mark all employees earning more than 50,000 as High Salary."

Row Filtering with Conditions

```
df[df["Age"] > 30]                                # Employees older than 30
df[df["City"] == "Dubai"]                         # Employees in Dubai
df[(df["Age"] > 30) & (df["Salary"] > 50000)]     # Multiple conditions
 df[(df["City"].isin(["Lahore", "Sharjah"]))]       # Belonging to multiple
                                                               categories
```

& = AND

| = OR

~ = NOT

Note: *Always use parentheses around conditions.*

Creating New Columns with Conditions

Using `np.where()`:

```
import numpy as np
df["Status"] = np.where(df["Age"] >= 18, "Adult", "Child")
```

Using `.apply()` with custom function:

```
def mark_salary(s):
    if s > 60000:
        return "High"
    elif s > 40000:
        return "Medium"
```

```
    else:
        return "Low"

df["SalaryCategory"] = df["Salary"].apply(mark_salary)
```

Using `.assign()` (inline column creation):

```
df = df.assign(Senior = df["Age"] > 40)
```

Conditional Replacement

`.where()` → Keeps values that meet condition, replaces others.

`.mask()` → Replaces values that meet condition.

```
df["Bonus"] = df["Bonus"].where(df["Bonus"] > 0, np.nan)          # Keep only positive
df["Salary"] = df["Salary"].mask(df["Salary"] < 0, 0)           # Replace negatives with 0
```

2. Grouping & Aggregation

Why Group Data?

- Summarise data by categories.
- Spot patterns (e.g., average salary by department).
- Similar to Excel pivot tables or SQL GROUP BY.

Basic Grouping

```
df.groupby("Department")["Salary"].mean()
```

Returns the average salary per department.

c) Multiple Aggregations with `agg()`

```
df.groupby("Department").agg(
    Avg_Salary=("Salary", "mean"),
    Max_Age=("Age", "max"),
    Employee_Count=("ID", "count")
)
```

Outputs a table with multiple summaries.

Group by Multiple Columns

```
df.groupby(["Department", "Gender"])["Salary"].mean()
```

Useful for cross-analysis (e.g., gender pay gap by department).

Sorting Group Results

```
df.groupby("Department")["Salary"].mean().sort_values(ascending=False)
```

Helps find top-performing departments quickly.

Pivot Tables

Pandas pivot tables mimic Excel pivot tables but with more flexibility.

```
pd.pivot_table(df,
               values="Salary",
               index="Department",
               columns="Gender",
               aggfunc="mean")
```

Produces a matrix of average salary by Department vs Gender.

Crosstab (Quick Counts)

```
pd.crosstab(df["Department"], df["Gender"])
```

Shows number of employees per Department × Gender.

Common Aggregation Functions

`mean()` → Average
`sum()` → Total
`count()` → Count of non-null
`nunique()` → Number of unique values
`max()`, `min()` → Highest/lowest
`median()` → Middle value

3. Data Visualisation

Why Visualise?

- Humans understand patterns faster in visuals.
- Helps explain insights to non-technical people.
- Detects outliers, trends, relationships.

Pandas Built-in Plotting (`.plot()`)

Pandas is built on Matplotlib, so `.plot()` is a shortcut.

```
# Histogram
df["Age"].plot(kind="hist", bins=10, title="Age Distribution")

# Line chart
df["Salary"].plot(kind="line", title="Salary Trend")

# Bar chart
df["Department"].value_counts().plot(kind="bar", title="Employees per
Department")

plt.show()
```

Matplotlib for Customisation

```
import matplotlib.pyplot as plt

plt.figure(figsize=(8,5))
plt.bar(df["Department"], df["Salary"])
plt.title("Department vs Salary")
plt.xlabel("Department")
plt.ylabel("Salary")
plt.show()
```

Seaborn for Advanced Visuals

Seaborn is built on Matplotlib but prettier by default.

```
import seaborn as sns

# Boxplot → See spread + outliers
sns.boxplot(x="Department", y="Salary", data=df)

# Countplot → Frequency of categories
sns.countplot(x="City", data=df)

# Scatterplot → Relationship between two variables
sns.scatterplot(x="Age", y="Salary", data=df)
```

When to Use Which Chart?

Histogram → Distribution of a single column

Boxplot → Spot outliers, check spread

Bar Chart → Compare categories

Line Chart → Show trends over time

Scatter Plot → Show relationship between two numeric variables

Customising Seaborn Charts

```
plt.figure(figsize=(8,6))
sns.barplot(x="Department", y="Salary", data=df, estimator=np.mean, ci=None)
plt.title("Average Salary by Department")
plt.xlabel("Department")
plt.ylabel("Salary")
plt.show()
```

- `estimator=np.mean` → Shows averages
- `ci=None` → Removes confidence intervals for cleaner look

Multiple Plots in One Go

```
fig, ax = plt.subplots(1,2, figsize=(12,5))
sns.histplot(df["Age"], bins=10, ax=ax[0])
sns.boxplot(x="Department", y="Salary", data=df, ax=ax[1])
plt.show()
```

Note: *Great for comparing distributions side by side.*

4. Practical Mini-Workflow

1. **Conditional**: Mark employees with "High" or "Low" salaries.
2. **Groupby**: Find average salary per department.
3. **Pivot**: Build a Department × Gender salary table.
4. **Visualise**:
   - Histogram → Age distribution.
   - Boxplot → Salary spread & outliers.
   - Bar chart → Avg salary per department.