# Azure Data Factory Data Flows & Advanced Concepts – Hands-on Labs

## Lab 1: Mapping Data Flows & Wrangling Data Flows

### Objective:

Understand how to build Mapping Data Flows and Wrangling Data Flows in ADF.

### Steps:

1. Open **ADF Studio** → **Author** hub.
2. Create a new **Mapping Data Flow**:
   * Add source dataset Employee\_CSV\_Dataset.
   * Add sink dataset Employee\_Transformed\_SQL.
   * Apply transformations (covered in next labs).
3. Create a **Wrangling Data Flow**:
   * Select source dataset (CSV or JSON).
   * Use Power Query editor to clean and shape data (remove nulls, rename columns).
   * Connect to a SQL sink dataset.
4. Debug and publish.

## Lab 2: Transformation Types – Join, Aggregate, Conditional Split, Derived Column

### Objective:

Practice common transformations in Mapping Data Flows.

### Steps:

1. Add a **Join** transformation:
   * Source1: Employee dataset.
   * Source2: Department dataset.
   * Join on DepartmentID.
2. Add an **Aggregate** transformation:
   * Group by DepartmentName.
   * Aggregate: Avg(Salary), Count(EmployeeID).
3. Add a **Conditional Split** transformation:
   * Condition: Salary > 100000 → HighSalary.
   * Else → RegularSalary.
4. Add a **Derived Column** transformation:
   * Create column AnnualBonus = Salary \* 0.10.

## Lab 3: Parameterization in Pipelines & Datasets

### Objective:

Pass dynamic values to datasets and pipelines.

### Steps:

1. Create a pipeline parameter FileName.
2. In dataset, create parameter FileNameParam.
3. Use expression: @dataset().FileNameParam in file path.
4. Map pipeline parameter to dataset parameter in Copy Activity.
5. Trigger pipeline with different file names to validate.

## Lab 4: Variables & Expressions

### Objective:

Use variables and expressions for dynamic runtime values.

### Steps:

1. Create a pipeline variable RowCount.
2. Add a **Set Variable** activity:
   * Expression: @activity('Lookup1').output.count.
3. Use expressions:
   * concat('Processed\_', pipeline().parameters.FileName).
   * formatDateTime(utcNow(), 'yyyy-MM-dd').

## Lab 5: Control Flow Activities – If, ForEach, Until

### Objective:

Implement control flow logic in pipelines.

### Steps:

1. Add an **If Condition** activity:
   * Expression: @equals(variables('RowCount'), 0).
   * True: Send notification, False: Continue.
2. Add a **ForEach** activity:
   * Items: List of files from Get Metadata.
   * Inside: Copy Activity to process each file.
3. Add an **Until** activity:
   * Condition: @greaterOrEquals(variables('RetryCount'),3).
   * Activities inside: Retry failed copy with incrementing counter.

## Lab 6: Pipeline Triggers – Schedule, Tumbling Window, Event-based

### Objective:

Automate pipeline execution using triggers.

### Steps:

1. **Schedule Trigger**:
   * Run every day at 2 AM.
2. **Tumbling Window Trigger**:
   * Interval = 1 hour.
   * Window dependency = true.
3. **Event-based Trigger**:
   * Storage Event = New file in Blob container.
   * Run pipeline automatically on new file arrival.

## Lab 7: Incremental Data Load Strategies

### Objective:

Implement incremental data loads using watermark.

### Steps:

1. Source SQL table has LastModifiedDate column.
2. Create pipeline parameter LastWatermark.
3. SQL query:

* SELECT \* FROM Employee WHERE LastModifiedDate > '@{pipeline().parameters.LastWatermark}'

1. Copy incremental data to ADLS.
2. Update watermark after successful load.

## Lab 8: Git Integration for CI/CD

### Objective:

Connect ADF to Git repo and commit changes.

### Steps:

1. In ADF Studio → **Manage** hub → Git Configuration.
2. Select Azure DevOps or GitHub.
3. Provide repo URL, collaboration branch, and folder path.
4. Save configuration.
5. Make pipeline changes → **Save All** → Commit with message.
6. Verify commit in repository.

## Lab 9: Logging & Monitoring Best Practices

### Objective:

Monitor pipelines and configure alerts.

### Steps:

1. Go to **Monitor** hub.
2. Check pipeline run history, activity run details.
3. Enable **Diagnostic Settings**:
   * Send logs to Log Analytics.
   * Export metrics to Azure Monitor.
4. Configure alerts for failed runs.
5. Implement retry policies on critical activities.