## **🔰 Week 0: Bootcamp Kickoff & Database Setup**

### **💡 Goals**

* Understand the data engineering landscape.
* Install Postgres with Docker.
* Connect using pgAdmin.

### **📌 Core Concepts**

| **Concept** | **Explanation** |
| --- | --- |
| **Data Engineer** | A plumber of the digital world: routes, cleans, and maintains data pipelines. |
| **Database** | Like a big Excel workbook but much more powerful and organized. |
| **PostgreSQL** | Open-source database that stores data in tables (like Excel sheets). |
| **Docker** | A lightweight virtual machine to run software in isolated environments. |

### **🧱 Setup Architecture**

Your Laptop

└── Docker

├── PostgreSQL Container (db)

└── pgAdmin Container (UI to view/edit db)

### **🔧 Step-by-Step Setup Lab**

#### **🧪 LAB 0.1: Run Postgres in Docker**

docker network create data-network

docker run --name postgres \

--network data-network \

-e POSTGRES\_PASSWORD=admin123 \

-e POSTGRES\_USER=admin \

-e POSTGRES\_DB=zachdb \

-p 5432:5432 -d postgres

#### **🧪 LAB 0.2: Launch pgAdmin**

docker run --name pgadmin \

--network data-network \

-e PGADMIN\_DEFAULT\_EMAIL=admin@zach.com \

-e PGADMIN\_DEFAULT\_PASSWORD=admin123 \

-p 5050:80 -d dpage/pgadmin4

💬 Visit<http://localhost:5050> and login with email + password.  
 Add server: name local, host postgres, user admin, db zachdb, password admin123.

## **📦 Week 1: Dimensional Data Modeling**

### **🧠 What is Dimensional Modeling?**

Dimensional modeling is like organizing your closet:

* **Fact Table**: central shelf with daily-used items (quantities, counts).
* **Dimension Tables**: side drawers describing your items (color, brand, date).

### **🔧 Core Concepts**

| **Term** | **Meaning** | **Example** |
| --- | --- | --- |
| **Fact Table** | Measures/events | Sales, Page Views |
| **Dimension Table** | Describes the facts | Product, User, Date |
| **SCD (Slowly Changing Dimension)** | Track changes in dimension values | User's last name or address |
| **Cumulative Table** | Aggregated over time | Daily Revenue to Date |
| **Graph Data Model** | Nodes and relationships | Facebook friends |

### **🎓 Real-World Example: Sales Database**

#### **Tables**

* fact\_sales: order\_id, product\_id, quantity, revenue, date\_id
* dim\_product: product\_id, name, category
* dim\_date: date\_id, day, month, year

### **🛠️ Lab Activities**

#### **🧪 LAB 1.1: Create Tables in Postgres**

sql

CopyEdit

CREATE TABLE dim\_product (

product\_id SERIAL PRIMARY KEY,

name TEXT,

category TEXT

);

CREATE TABLE dim\_date (

date\_id DATE PRIMARY KEY,

day INT,

month INT,

year INT

);

CREATE TABLE fact\_sales (

order\_id SERIAL PRIMARY KEY,

product\_id INT REFERENCES dim\_product(product\_id),

date\_id DATE REFERENCES dim\_date(date\_id),

quantity INT,

revenue NUMERIC

);

#### **🧪 LAB 1.2: Insert Sample Data**

sql

INSERT INTO dim\_product (name, category) VALUES

('Shampoo', 'Health'), ('Sofa', 'Furniture');

INSERT INTO dim\_date VALUES

('2023-01-01', 1, 1, 2023), ('2023-01-02', 2, 1, 2023);

INSERT INTO fact\_sales (product\_id, date\_id, quantity, revenue) VALUES

(1, '2023-01-01', 2, 10.99), (2, '2023-01-02', 1, 399.99);

### **🔁 LAB 1.3: Slowly Changing Dimension (SCD Type 2)**

Track user address changes by versioning:

CREATE TABLE dim\_customer\_scd (

customer\_id INT,

name TEXT,

address TEXT,

effective\_from DATE,

effective\_to DATE,

is\_current BOOLEAN

);

Insert new rows instead of updating old ones, and set is\_current = false on old rows.

### **🧭 Zach’s Wisdom**

* **Draw ERDs** before coding. Use dbdiagram.io or draw.io.
* Model **what questions you want answered** from the business.
* If your facts change slowly, it’s a dimension in disguise!

## **📦 Week 2: Fact Data Modeling (Facebook-style Advanced)**

### **🎯 Week 2 Learning Goals**

* Understand different types of fact tables (transactional, snapshot, accumulating).
* Model large-scale event data using the **Facebook-style datelist** trick.
* Learn how to reduce **shuffle** in Spark using **long array metrics**.
* Start thinking like a scalable data engineer.

## **🔍 1. What Are Fact Tables?**

| **Type of Fact Table** | **Description** | **Example** |
| --- | --- | --- |
| **Transactional** | One row per event | fact\_sales, fact\_clicks |
| **Snapshot** | Periodic record of state | Daily active users |
| **Accumulating** | Grows over time as stages complete | Order lifecycle (ordered → shipped → delivered) |

## **🧠 Facebook’s Secret Sauce: The datelist Table**

Let’s say you want to answer:

“How many users were active each day over the past 30 days?”

But user activity is **sparse**. Instead of storing one row per user per day (expensive!), you use:

user\_id | active\_dates

--------|----------------------

u001 | [2023-01-01, 2023-01-03, 2023-01-07]

u002 | [2023-01-02, 2023-01-03]

This is called a **long array metric**.

### **✅ Benefits:**

* Compresses sparse events.
* Easy to explode later in Spark or SQL.
* Optimized for read-heavy analytical workflows.

## **🏗️ 2. Schema: User Activity with Long Arrays**

sql

CopyEdit

CREATE TABLE fact\_user\_activity (

user\_id TEXT PRIMARY KEY,

active\_dates DATE[]

);

This is PostgreSQL. In Spark, you'll use ArrayType(DateType).

## **🔬 LAB 2.1: Populate a Long Array Metric Table**

sql

CopyEdit

INSERT INTO fact\_user\_activity VALUES

('u001', ARRAY['2023-01-01', '2023-01-03', '2023-01-07']),

('u002', ARRAY['2023-01-02', '2023-01-03']);

#### **📊 LAB 2.2: Explode Arrays into Individual Events**

sql

CopyEdit

SELECT user\_id, unnest(active\_dates) AS activity\_date

FROM fact\_user\_activity;

Output:

| **user\_id** | **activity\_date** |
| --- | --- |
| u001 | 2023-01-01 |
| u001 | 2023-01-03 |
| u001 | 2023-01-07 |
| u002 | 2023-01-02 |
| u002 | 2023-01-03 |

## **⚙️ 3. Minimizing Shuffle in Spark**

### **Problem:**

You have billions of events. Joining and aggregating by user causes massive **shuffle**, which slows jobs down.

### **Solution:**

Pre-aggregate and store arrays:

* Track last 30 days of behavior in arrays.
* Only explode when necessary.
* Cache intermediate tables.

#### **⚡ Spark Trick:**

from pyspark.sql.functions import collect\_list, to\_date

activity\_df = raw\_logs \

.groupBy("user\_id") \

.agg(collect\_list(to\_date("event\_ts")).alias("active\_dates"))

## **📈 4. Fact Table Design Checklist**

✅ Includes foreign keys to all dimensions  
 ✅ Granularity is well defined (e.g., 1 row = 1 click)  
 ✅ Doesn’t store slowly changing info (that goes in dimension!)  
 ✅ Designed for partitioning & pruning (e.g., by date\_id)

## **🧪 LAB 2.3: Create a Facebook-Style Activity Dashboard**

### **1. Create supporting dimension:**

CREATE TABLE dim\_user (

user\_id TEXT PRIMARY KEY,

name TEXT

);

### **2. Join exploded activity with dim\_user:**

SELECT d.name, a.activity\_date

FROM fact\_user\_activity f

JOIN dim\_user d ON f.user\_id = d.user\_id,

unnest(f.active\_dates) a(activity\_date);

## **🎯 Zach’s Wisdom of the Week**

* "Facts are events. Don't model fluff — model things that happened."
* "Storage is cheap. Shuffle is expensive."
* "Pre-aggregate like you're Google. Explode like you're Amazon."

## **📦 Architecture View: Fact Table-Based Model**

+-----------------+

| dim\_user |

+-----------------+

|

| user\_id

▼

+-----------------------+

| fact\_user\_activity |

+-----------------------+

| user\_id |

| active\_dates (ARRAY) |

+-----------------------+

## **🧠 Quiz Yourself**

* What’s the difference between transactional and accumulating fact tables?
* When should you use a long array vs one row per event?
* How do you reduce shuffle in Spark?

# **⚡ Week 3: Apache Spark for Dummies**

**“Write Once. Scale Forever.” — Zach Wilson**

## **🎯 Week 3 Learning Goals**

* Understand **what Spark is** and why it's used.
* Learn **Spark transformations and actions**.
* Practice **partitioning and memory tuning**.
* Implement **unit and integration testing** in PySpark.

## **🚀 What is Apache Spark?**

Spark is like Excel… on steroids… for big data.

| **🔍 Feature** | **📖 Explanation** |
| --- | --- |
| **Distributed** | Runs on many machines in parallel |
| **In-Memory** | Processes data in memory (faster than disk-based like Hadoop) |
| **Scalable** | Handles small to petabyte-scale data |
| **Lazy Evaluation** | Doesn’t compute until you ask for results |

🧠 Think of Spark like a kitchen: you prep your ingredients (transformations), but nothing cooks (action) until you say “run”.

## **🧰 Spark Basics – Hands-on with PySpark**

### **🧱 Architecture**

Driver Program

├── SparkSession

└── Cluster Manager

├── Executor 1

├── Executor 2

└── Executor N

### **🔧 LAB 3.1: Your First Spark Job (Local Mode)**

from pyspark.sql import SparkSession

spark = SparkSession.builder \

.appName("Spark Basics") \

.getOrCreate()

data = [("Alice", 1), ("Bob", 2), ("Cathy", 3)]

df = spark.createDataFrame(data, ["name", "id"])

df.show()

## **🔄 Spark Transformations vs Actions**

| **💡 Transformations** | **💥 Actions** |
| --- | --- |
| map, filter, groupBy | count(), show(), collect() |
| lazy (doesn't run) | triggers actual computation |

df.filter(df["id"] > 1) # ← transformation

df.count() # ← action (triggers filter)

## **🧪 LAB 3.2: Real Use Case – Filter Logs**

logs = [

("2023-07-01", "INFO"),

("2023-07-02", "ERROR"),

("2023-07-03", "INFO")

]

df = spark.createDataFrame(logs, ["date", "level"])

df.filter(df["level"] == "ERROR").show()

## **🧠 Memory Tuning 101**

### **Problem:**

Your Spark job crashes or slows down.

### **Solution:**

Control the following:

* spark.executor.memory – how much RAM each task can use
* spark.sql.shuffle.partitions – default is 200; lower it for small data
* .repartition() – helps control shuffles
* .persist() – keep intermediate data in memory

df = df.repartition(4).persist()

## **🧪 LAB 3.3: Partitioning**

df = spark.range(0, 20).repartition(4)

print(df.rdd.getNumPartitions()) # should be 4

## **🧪 LAB 3.4: Caching Intermediate Results**

transformed = df.filter(df["id"] > 5).persist()

transformed.count()

transformed.show()

Without persist(), Spark recomputes each time. With it, Spark stores it in memory.

## **✅ Unit Testing Spark Jobs**

We don’t just write data pipelines — we test them like pros.

### **🧪 LAB 3.5: Pytest + Spark Unit Test**

Create test\_pipeline.py:

from pyspark.sql import SparkSession

def test\_uppercase\_names():

spark = SparkSession.builder.master("local").appName("Test").getOrCreate()

df = spark.createDataFrame([("alice",)], ["name"])

df2 = df.withColumn("name", upper(df["name"]))

result = df2.collect()[0]["name"]

assert result == "ALICE"

Run with pytest test\_pipeline.py

## **🧪 Integration Testing = Real Data Sample**

Use a **.csv** file or mock database to test the full pipeline.

df = spark.read.csv("data/sample.csv", header=True, inferSchema=True)

assert df.count() > 0

## **🎯 Spark Cheat Sheet**

| **Concept** | **Code Snippet** |
| --- | --- |
| Read CSV | spark.read.csv() |
| Filter | df.filter(df["col"] == "x") |
| GroupBy | df.groupBy("col").count() |
| Repartition | df.repartition(4) |
| Persist | df.persist() |
| Write Parquet | df.write.parquet("path") |

## **🧠 Zach’s Wisdom of the Week**

* “If it doesn’t fit in memory, partition it. If it runs twice, persist it.”
* “Spark won’t fix bad modeling. Garbage in, distributed garbage out.”
* “Unit test your transformations. Integration test your logic. Monitor everything.”

## **💡 Real-World Use Case**

You’re working at Netflix and want to track:

* Total hours watched per user
* Filter out spam devices
* Cache intermediate data for week-over-week analysis

## **📈 Architecture View – Spark in Data Pipeline**

Kafka Stream / File Ingest

↓

Spark ETL Job

↓

Intermediate Parquet Files

↓

Snowflake / Postgres

↓

Tableau / Power BI

### **📎 Homework**

* Write 2 PySpark transformations with filter and groupBy
* Create a small test with pytest and Spark
* Optimize a join or transformation with .persist() or .repartition()

# **🌊 Week 4: Real-Time Data Engineering with Kafka + Flink**

**“Stream like Netflix. Buffer like a boss.” — Zach Wilson**

## **🎯 Week 4 Learning Goals**

* Understand **Apache Kafka** (the backbone of modern data streams).
* Learn **Apache Flink** (real-time transformation and computation engine).
* Implement **window functions** to handle time-based aggregations.
* Build your first **end-to-end real-time streaming pipeline**.

## **🧠 What Is Kafka?**

Kafka is like a **giant message board** where different apps post and read updates in real-time.

| **Term** | **Meaning** |
| --- | --- |
| **Producer** | App that sends messages to Kafka |
| **Consumer** | App that reads messages |
| **Topic** | Named stream/channel |
| **Partition** | Split of a topic for scaling |
| **Broker** | Kafka server that stores messages |

## **🧱 Kafka Architecture**

sql

CopyEdit

[ Producer ] --> [ Kafka Topic ] --> [ Consumer ]

|

+----------+

|Partition 1|

|Partition 2|

+----------+

## **🛠️ LAB 4.1: Kafka with Docker (Zookeeper + Broker)**

**🧰 Tools Needed**: Docker Desktop, VS Code

**📁 Folder Structure:**

sql

CopyEdit

real-time-project/

├── docker-compose.yml

**docker-compose.yml**

version: '2'

services:

zookeeper:

image: confluentinc/cp-zookeeper:7.2.1

environment:

ZOOKEEPER\_CLIENT\_PORT: 2181

kafka:

image: confluentinc/cp-kafka:7.2.1

depends\_on:

- zookeeper

ports:

- "9092:9092"

environment:

KAFKA\_BROKER\_ID: 1

KAFKA\_ZOOKEEPER\_CONNECT: zookeeper:2181

KAFKA\_ADVERTISED\_LISTENERS: PLAINTEXT://localhost:9092

KAFKA\_OFFSETS\_TOPIC\_REPLICATION\_FACTOR: 1

Run with:

bash

CopyEdit

docker-compose up -d

## **✅ LAB 4.2: Kafka CLI — Create Topic and Send Data**

docker exec -it kafka bash

# Create topic

kafka-topics --create --topic user\_events --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1

# Send messages

kafka-console-producer --topic user\_events --bootstrap-server localhost:9092

👉 Type:

{"user\_id": "u001", "action": "login", "ts": "2025-07-07T08:00:00"}

## **🧠 What is Apache Flink?**

Flink is a **real-time SQL + Python engine** built for streaming data and windowed processing.

| **Feature** | **Meaning** |
| --- | --- |
| **Event Time** | Timestamp when event actually occurred |
| **Windowing** | Group data into time buckets |
| **Watermarking** | Handles late data |
| **Sink** | Where the results go (DB, file, etc.) |

## **🔧 LAB 4.3: Run Flink in Docker**

bash

CopyEdit

docker run -d -p 8081:8081 flink

Visit:<http://localhost:8081>

## **🔄 Flink Job Flow**

scss

CopyEdit

Kafka (user\_events topic)

↓

Flink Job (filter + groupBy + window)

↓

Output sink (print, Postgres, S3)

## **🧪 LAB 4.4: Python Flink App (PyFlink)**

**streaming\_app.py**

python

from pyflink.datastream import StreamExecutionEnvironment

from pyflink.datastream.connectors import FlinkKafkaConsumer

from pyflink.common.serialization import SimpleStringSchema

env = StreamExecutionEnvironment.get\_execution\_environment()

kafka\_props = {

'bootstrap.servers': 'localhost:9092',

'group.id': 'test'

}

consumer = FlinkKafkaConsumer(

topics='user\_events',

deserialization\_schema=SimpleStringSchema(),

properties=kafka\_props

)

ds = env.add\_source(consumer)

ds.print()

env.execute("Kafka to Flink Stream")

## **⏳ What Are Windows in Flink?**

| **Window Type** | **Use Case** |
| --- | --- |
| **Tumbling** | Fixed time window (e.g., every 5 min) |
| **Sliding** | Overlapping windows (e.g., every 1 min over 5-min windows) |
| **Session** | Gaps between user activity |

### **🧪 LAB 4.5: Tumbling Window – Count Logins Per Minute**

python

ds \

.key\_by(lambda x: json.loads(x)["user\_id"]) \

.window(TumblingEventTimeWindows.of(Time.minutes(1))) \

.reduce(lambda a, b: a + 1)

You can write the results to print, database, or file using .add\_sink().

## **✅ Real-World Use Case**

Imagine you’re at **Uber**:

* Producer: Mobile apps push trip status in real time.
* Kafka Topic: trip\_events
* Flink: Detect surge pricing zones every 2 mins using sliding windows.
* Sink: Publish metrics to Redis for live map display.

## **📈 Architecture View – Kafka + Flink Streaming**

css

[ Mobile App ] ➝ Kafka ➝ Flink ➝ PostgreSQL ➝ Power BI Dashboard

## **📎 Homework**

* Run Kafka locally and send 5 events to a topic
* Set up Flink (or PyFlink) to consume from Kafka
* Implement tumbling and sliding windows with JSON logs

## **🧠 Zach’s Wisdom of the Week**

* "Real-time systems = faster insights = faster business decisions."
* "If batch is a train, streaming is an Uber — always running, always live."
* "Windowing is to streams what GROUP BY is to SQL."

# **🛡️ Week 5: Data Quality + Contracts + Analytical Patterns**

**"A pipeline without tests is a lawsuit waiting to happen." — Zach Wilson**

## **🎯 Week 5 Learning Goals**

* Implement **data contracts** like Airbnb
* Build a **write → audit → publish** pipeline
* Track **funnel and growth metrics**
* Automate data quality checks with tools like **dbt tests** or Python

## **🔐 1. What Is a Data Contract?**

A **data contract** is like a *restaurant menu* — it guarantees what data is delivered, how it’s structured, and what to expect.

| **👀 Without Contract** | **✅ With Contract** |
| --- | --- |
| “Sometimes we get nulls” | “Field user\_id is always non-null and type STRING” |
| “Revenue format changed last week” | “Schema changes are version-controlled” |

## **🧱 Airbnb’s Model: Write → Audit → Publish**

Raw Layer (Write)

↓

Validation Layer (Audit)

↓

Business Layer (Publish)

### **🧪 LAB 5.1: dbt Model with Contracts (YAML + SQL)**

📁 models/user\_metrics.sql

sql

CopyEdit

SELECT

user\_id,

signup\_date,

last\_active\_date

FROM raw\_users

📁 models/user\_metrics.yml

yaml

version: 2

models:

- name: user\_metrics

description: Cleaned user activity table

columns:

- name: user\_id

tests:

- not\_null

- unique

- name: signup\_date

tests:

- not\_null

Run with:

bash

CopyEdit

dbt run

dbt test

✅ This checks data **before** it reaches stakeholders.

## **⚠️ Common Quality Tests**

| **Test** | **Purpose** |
| --- | --- |
| not\_null | Ensure no blanks in critical columns |
| unique | Prevent duplicate IDs |
| accepted\_values | Check values in enum-like fields |
| relationships | Validate foreign keys |

## **📊 2. Analytical Pattern: Funnel Analysis**

Funnel = sequence of events leading to conversion (like **signup → onboarding → purchase**)

### **🧪 LAB 5.2: Funnel in SQL (Daily Counts)**

sql

WITH funnel AS (

SELECT

user\_id,

MIN(CASE WHEN event = 'signup' THEN event\_time END) AS signup\_time,

MIN(CASE WHEN event = 'onboarding' THEN event\_time END) AS onboard\_time,

MIN(CASE WHEN event = 'purchase' THEN event\_time END) AS purchase\_time

FROM events

GROUP BY user\_id

)

SELECT

COUNT(\*) AS started,

COUNT(onboard\_time) AS onboarded,

COUNT(purchase\_time) AS purchased

FROM funnel;

## **📈 3. Analytical Pattern: Growth Accounting**

Answer: “Where is growth coming from?” (New, returning, resurrected users)

### **🧪 LAB 5.3: User Classification Over Time**

sql

WITH daily\_users AS (

SELECT user\_id, DATE(event\_time) AS event\_date

FROM events

GROUP BY user\_id, DATE(event\_time)

),

classify AS (

SELECT

user\_id,

event\_date,

MIN(event\_date) OVER (PARTITION BY user\_id) AS first\_seen

FROM daily\_users

)

SELECT

event\_date,

COUNT(CASE WHEN event\_date = first\_seen THEN 1 END) AS new\_users,

COUNT(CASE WHEN event\_date > first\_seen THEN 1 END) AS returning\_users

FROM classify

GROUP BY event\_date;

## **🧰 Extra Credit: Implement Contracts in JSON Schema**

json

{

"title": "User Metrics Schema",

"type": "object",

"properties": {

"user\_id": { "type": "string" },

"signup\_date": { "type": "string", "format": "date" }

},

"required": ["user\_id", "signup\_date"]

}

This contract can be versioned and validated in CI/CD pipelines using tools like **Great Expectations** or **Pandera**.

## **🔐 Real-World Example: Netflix**

* Write: Raw user clicks
* Audit: Validate user\_id + action enums
* Publish: Clean data to Tableau
* Contracts: Schema registry to define what’s expected

## **🧠 Zach’s Wisdom of the Week**

* “Data breaks. Contracts protect your credibility.”
* “Always test BEFORE dashboards, not after someone yells.”
* “Funnel = your user’s journey. Growth accounting = your investor’s report.”

## **📈 Architecture View – Data Quality Pipeline**

css

[ Raw Kafka/Postgres ]

↓

[ dbt (contract + test layer) ]

↓

[ Curated Tables → Metrics Dashboards ]

## **✅ Homework**

* Add not\_null and unique tests to 1 dbt model
* Build a SQL funnel query with 3 event stages
* Add growth accounting over past 7 days in SQL

# **🎤 Week 6: Communicating Impact with KPIs, Dashboards & Experimentation**

**“Your pipeline isn’t finished until your stakeholder understands the chart.” — Zach Wilson**

## **🎯 Week 6 Learning Goals**

* Present your work through **impactful dashboards**.
* Define and track **KPIs** (Key Performance Indicators).
* Use tools like **Tableau**, **Power BI**, or **Streamlit**.
* Communicate **A/B testing results** and data stories.
* Maintain and monitor pipelines like a pro.

## **📌 1. Why Communication Matters**

You don’t just build pipelines. You **translate data into business decisions**.

| **👎 Bad** | **👍 Great** |
| --- | --- |
| “We processed 2 TB of data.” | “We identified a 15% drop in checkout conversion.” |
| “I fixed a dbt job.” | “This fix saved 20 hours of manual QA per week.” |

## **📈 2. KPI Framework (Zach Style)**

**Think like a Product Manager:**

| **Metric Type** | **Example** |
| --- | --- |
| **Input KPIs** | Signups, Visits, Searches |
| **Output KPIs** | Purchases, Revenue, Retention |
| **Health KPIs** | Latency, Errors, Uptime |

✅ Use a dashboard layout that starts with high-level KPIs and drills down to root cause.

## **🛠️ LAB 6.1: Build a Power BI Dashboard (Healthcare Example)**

📁 Dataset: hospital\_metrics.csv  
 Columns: hospital\_id, admissions, avg\_stay, billing\_amount, date

### **Steps:**

1. Load data in Power BI
2. Create Date Dimension Table

Create DAX Measures:  
  
 DAX  
CopyEdit  
Total Admissions = SUM(hospital\_metrics[admissions])

Avg Stay = AVERAGE(hospital\_metrics[avg\_stay])

1. Create KPIs:  
   * Total Billing
   * MoM Change in Avg Stay
   * Top 5 Hospitals by Revenue
2. Add filters: Hospital, Date Range

## **🧪 LAB 6.2: A/B Test Dashboard (Growth Experiment)**

### **Table: experiment\_results**

| **group** | **conversions** | **visitors** |
| --- | --- | --- |
| control | 200 | 1000 |
| variant | 260 | 1000 |

### **KPI: Conversion Rate = conversions / visitors**

Use this to calculate lift:

sql

CopyEdit

SELECT

group,

conversions \* 1.0 / visitors AS conversion\_rate

FROM experiment\_results;

Plot:

* Conversion rate per group
* Lift (% increase)
* Significance (Z-score or p-value)

Use **Statsig**, **Optimizely**, or manual hypothesis testing.

## **📊 3. Storytelling With Data – The Zach Framework**

"Don’t show dashboards. Tell stories."

Use this template:

1. **Goal**: What did the business want to improve?
2. **Observation**: What data did you find?
3. **Action**: What change was made?
4. **Impact**: What business result was achieved?

## **🔁 4. Maintenance: Pipelines, Alerts, and Dashboards**

| **Task** | **Tool** |
| --- | --- |
| Pipeline monitoring | Airflow, Prefect |
| Dashboard alerts | Power BI subscriptions, email triggers |
| Regression testing | dbt tests, data diff tools |

## **✅ Real-World Example: Shopify Growth Team**

* KPI Dashboard tracks new merchants, 7-day retention, revenue per cohort.
* Each team owns a dbt model + Power BI tab.
* Weekly review includes:  
  + Wins
  + Blockers
  + KPIs up/down + action plan

## **🧠 Zach’s Wisdom of the Week**

* “No one cares how fast your Spark job runs unless it moves a KPI.”
* “Visualize like a designer, speak like a CEO.”
* “Your impact = business dollars saved or earned.”

## **🧰 Architecture: Full Data-to-Impact Flow**

[Kafka / API / S3]

↓

[Spark/Flink Streaming]

↓

[dbt with Contracts]

↓

[Snowflake / Postgres DB]

↓

[Power BI / Tableau / Streamlit]

↓

[Stakeholders Making Decisions 🚀]

## **✅ Final Homework**

* Build a 1-page KPI dashboard with 3 metrics
* Create a growth experiment summary (A/B results)
* Share 1 data story with a stakeholder or mentor

## **🎓 Graduation: Data Engineer 1.0 to 2.0**

Congratulations, You’ve just walked through **Zach Wilson’s 6-Week Data Engineering Bootcamp** — dummified, distilled, and delivered with labs, architecture, and impact in mind.

## **🧳 What’s Next?**

1. **Capstone Project Ideas**
2. Real-time streaming for hospital alerts
3. Marketing attribution using dbt + Snowflake
4. Agentic AI dashboard with LangChain + Streamlit