16 DATA ENGINEERING PROJECTS

The 6 from the video plus 10 more, every one built with free tools. Each project gives you:

+ What it proves to an employer, the time and the level
+ The tools, with links
+ Step-by-step build with the exact code and commands
+ How to check it works, and what you should see
+ What to do when it breaks
+ An interview question you can now answer
+ A line for your resume

START HERE

Install Python and Docker Desktop. Those two cover most of the 16 projects. Then do project 1 today: it takes about two hours.

python.org/downloads docker.com/products/docker-desktop

THE 16 PROJECTS

Pick one, build it, post it

Tap a project to jump to it. Every link in this guide is tappable.

#	Project	Time	Level	Main tool
01	Batch ETL: API to Postgres	2 hours	Beginner	Python 3.12
02	Cloud Warehouse in BigQuery	1.5 hours	Beginner	BigQuery sandbox
03	NYC Taxi Trip Analysis	2 hours	Beginner	DuckDB
04	Real-Time Click Stream	3 hours	Intermediate	Apache Kafka
05	Star Schema with dbt	3 hours	Intermediate	dbt Core + dbt-duckdb
06	Daily Pipeline on Airflow	3 hours	Intermediate	Apache Airflow
07	Partitioned Data Lake on Your Laptop	1.5 hours	Beginner	DuckDB
08	Incremental Load with a Watermark	2 hours	Intermediate	PostgreSQL 16
09	Customer History with dbt Snapshots	1 hour	Intermediate	dbt Core + dbt-duckdb
10	Data Quality Report in SQL	1.5 hours	Beginner	DuckDB + Python
11	Spark Batch Job	2 hours	Intermediate	PySpark
12	Scheduled Pipeline on GitHub Actions	1 hour	Beginner	GitHub Actions
13	Data API with FastAPI	1.5 hours	Intermediate	FastAPI + Uvicorn
14	Self-Hosted Dashboard with Metabase	1 hour	Beginner	Metabase (open source)
15	Web Log Pipeline	2 hours	Intermediate	Python + pandas + pyarrow
16	End-to-End Capstone	1 to 2 weekends	Advanced	Airflow

Projects 1 to 6 are the ones from the video. 7 to 16 are the next level.

HOW TO SHOW IT ON YOUR RESUME

One GitHub repo per project, with this layout. Recruiters and interviewers open the README first.

```
project-01-batch-etl/
  README.md        what you built, why, the diagram
  src/             your pipeline code
  sql/             table definitions and queries
  screenshots/     the "check it works" results
  LESSONS.md       what broke and how you fixed it
```

Then put the resume line from the end of each project on your resume and LinkedIn, with the repo link.

Batch ETL: API to Postgres

You can pull data from a real API, clean it, and load it into a database so a rerun never creates duplicates.

TIME 2 hours	LEVEL Beginner

TOOLS

Python 3.12 requests, pandas, SQLAlchemy

Docker Desktop free for personal use

PostgreSQL 16 official Docker image

Open-Meteo API free for non-commercial use, no API key

WHAT YOU BUILD

Piece	What it is	Where
Source	Daily weather forecast, JSON	api.open-meteo.com/v1/forecast
Code	etl.py: extract, transform, load	your project folder
Database	Postgres 16 in Docker	localhost:5432, user postgres
Table	weather_daily (city, day, temp_max, temp_min, precip_mm)	primary key (city, day)

BUILD IT

01 Start Postgres in Docker.

One command gives you a real database on your laptop.

```
docker run -d --name pg -e POSTGRES_PASSWORD=pass \
  -p 5432:5432 postgres:16
```

02 Set up Python.

A virtual environment keeps this project's packages separate. On Windows activate with .venv\Scripts\activate.

```
python -m venv .venv
source .venv/bin/activate
pip install requests pandas sqlalchemy "psycopg[binary]"
```

03 Create the table.

Open psql inside the container and create the target table. The primary key is what makes reruns safe.

```
docker exec -it pg psql -U postgres

CREATE TABLE weather_daily (
  city text, day date,
  temp_max real, temp_min real, precip_mm real,
  PRIMARY KEY (city, day));
```

04 Extract.

Call the API. It returns 7 days of forecast by default.

```
import requests, pandas as pd
url = "https://api.open-meteo.com/v1/forecast"
params = {"latitude": 40.71, "longitude": -74.01,
  "daily": "temperature_2m_max,temperature_2m_min,precipitation_sum",
  "timezone": "auto"}
daily = requests.get(url, params=params, timeout=30).json()["daily"]
```

05 Transform.

Give every column a name and a type, and drop rows with no reading.

```
df = pd.DataFrame({
  "city": "New York",
  "day": pd.to_datetime(daily["time"]).date,
  "temp_max": daily["temperature_2m_max"],
  "temp_min": daily["temperature_2m_min"],
  "precip_mm": daily["precipitation_sum"]})
df = df.dropna(subset=["temp_max"])
```

06 Load with an upsert.

INSERT ... ON CONFLICT updates a day that is already there instead of adding it twice.

```
from sqlalchemy import create_engine, text
engine = create_engine(
  "postgresql+psycopg://postgres:pass@localhost:5432/postgres")
sql = text("""INSERT INTO weather_daily
  VALUES (:city, :day, :temp_max, :temp_min, :precip_mm)
  ON CONFLICT (city, day) DO UPDATE SET
  temp_max = EXCLUDED.temp_max, temp_min = EXCLUDED.temp_min,
  precip_mm = EXCLUDED.precip_mm""")
with engine.begin() as conn:
    conn.execute(sql, df.to_dict("records"))
print(len(df), "rows loaded")
```

07 Run it twice.

The second run is the test. The row count must not change.

```
python etl.py
python etl.py
```

CHECK IT WORKS

Run or check	You should see
`SELECT COUNT(*) FROM weather_daily;`	7 after the first run, still 7 after the second
`SELECT * FROM weather_daily ORDER BY day;`	one row per day, temperatures filled in
`docker ps`	the pg container listed as Up

IF IT BREAKS

Problem	Fix
connection refused	The container is not running. docker start pg, then retry.
No module named psycopg	pip install "psycopg[binary]" inside the activated .venv.
KeyError: 'daily'	Print the response. A misspelled variable name in params returns an error message instead of data.

GO FURTHER

Loop over five cities and write each run's row count and time to a load_log table.

INTERVIEW QUESTION YOU CAN NOW ANSWER

What does it mean for a pipeline to be idempotent?

Running it again with the same input gives the same result. Here the upsert on (city, day) means a rerun updates rows instead of duplicating them.

RESUME LINE

Built a Python ETL pipeline that extracts daily weather data from a REST API, cleans it with pandas and upserts it into PostgreSQL in Docker, idempotent on reruns.

Cloud Warehouse in BigQuery

You can build a warehouse table the way companies do: partitioned and clustered, so queries scan less data and cost less.

TIME 1.5 hours	LEVEL Beginner

TOOLS

BigQuery sandbox no credit card; 10 GB storage, 1 TB queries a month

Looker Studio free dashboards

WHAT YOU BUILD

Piece	What it is	Name
Project	Your sandbox project	any name you like
Source	NYC yellow taxi trips 2022, public	bigquery-public-data.new_york_taxi_trips
Dataset	Your own dataset, US location	taxi
Table	January trips, partitioned by day	taxi.trips_jan
View	Trips and revenue per day	taxi.daily_revenue

BUILD IT

01 Open the sandbox.

Go to console.cloud.google.com/bigquery with a Google account, accept the terms and create a project. A sandbox notice appears at the top. Tables expire after 60 days in the sandbox.

02 Create your dataset.

In the query editor. US matches where the public data lives. To practise with your own file instead, use Create table > Upload and pick a CSV.

```
CREATE SCHEMA taxi OPTIONS(location = "US");
```

03 Look before you copy.

Before running, read the estimate at the top right of the editor: it shows how many bytes the query will scan.

```
SELECT COUNT(*)
FROM `bigquery-public-data.new_york_taxi_trips.tlc_yellow_trips_2022`
WHERE DATE(pickup_datetime) BETWEEN '2022-01-01' AND '2022-01-31';
```

04 Build a partitioned, clustered table.

One month only, so you stay well inside the free storage.

```
CREATE TABLE taxi.trips_jan
PARTITION BY DATE(pickup_datetime)
CLUSTER BY pickup_location_id AS
SELECT pickup_datetime, dropoff_datetime, passenger_count,
  trip_distance, fare_amount, tip_amount, total_amount,
  pickup_location_id, dropoff_location_id
FROM `bigquery-public-data.new_york_taxi_trips.tlc_yellow_trips_2022`
WHERE DATE(pickup_datetime) BETWEEN '2022-01-01' AND '2022-01-31';
```

05 Compare bytes scanned.

Paste each query and read the estimate without running it. Screenshot both for your README.

```
-- A: everything
SELECT * FROM taxi.trips_jan;

-- B: one day, two columns
SELECT pickup_location_id, AVG(total_amount) AS avg_total
FROM taxi.trips_jan
WHERE DATE(pickup_datetime) = '2022-01-15'
GROUP BY 1;
```

06 Build a view for reporting.

```
CREATE VIEW taxi.daily_revenue AS
SELECT DATE(pickup_datetime) AS day, COUNT(*) AS trips,
  ROUND(SUM(total_amount), 2) AS revenue
FROM taxi.trips_jan
GROUP BY day;
```

07 Chart it in Looker Studio.

lookerstudio.google.com > Create > Report > BigQuery > your project > taxi > daily_revenue. Add a time series of revenue by day.

CHECK IT WORKS

Run or check	You should see
Table Details tab	Partitioned by DAY on pickup_datetime, clustered by pickup_location_id
Estimate for query B vs A	B scans a small fraction of A
SELECT COUNT(*) FROM taxi.daily_revenue;	31

IF IT BREAKS

Problem	Fix
Dataset location error	Create the dataset in US, the same location as the public data.
A feature asks for billing	The sandbox leaves out some features (such as streaming). Stick to queries, tables and views.
Table disappeared	Sandbox tables expire after 60 days. Rerun step 4.

GO FURTHER

Add February as a second table, then UNION the two in a view and filter by date to see pruning across both.

INTERVIEW QUESTION YOU CAN NOW ANSWER

Why partition and cluster a warehouse table?

Partitioning lets a query skip whole days it does not need. Clustering sorts data inside each partition so filters on that column read fewer blocks. BigQuery charges by bytes scanned, so both cut cost and time.

RESUME LINE

Built a partitioned, clustered BigQuery table from NYC taxi data, cut bytes scanned with partition filters, and published a daily revenue view to a Looker Studio dashboard.

PROJECT 03 / SQL & PYTHON / FROM THE VIDEO

NYC Taxi Trip Analysis

You can query millions of rows with SQL, move results into Python, and turn them into findings someone can read.

TIME 2 hours	LEVEL Beginner

TOOLS

DuckDB free, runs inside Python

pandas + matplotlib dataframes and charts

NYC TLC trip data free monthly Parquet files

WHAT YOU BUILD

File	What it is	Where
yellow_tripdata_2024-01.parquet	One month of yellow taxi trips	TLC page, 2024 > January > Yellow
taxi_zone_lookup.csv	Zone ID to borough and zone name	TLC page, Taxi Zone Lookup Table
analysis.py or a notebook	Your queries and charts	your project folder

BUILD IT

01 Install the tools.

```
pip install duckdb pandas matplotlib jupyter
```

02 Download the data.

From the TLC trip record page, download January 2024 Yellow Taxi Trip Records and the Taxi Zone Lookup Table into your project folder.

03 Count the rows straight from the file.

DuckDB reads Parquet directly. No loading step.

```
import duckdb
con = duckdb.connect()
T = "'yellow_tripdata_2024-01.parquet'"
con.sql(f"SELECT COUNT(*) FROM {T}").show()
```

04 Clean as you query.

Keep real trips in the right month. Use this filter in every query below.

```
WHERE trip_distance > 0 AND total_amount > 0
  AND tpep_pickup_datetime >= '2024-01-01'
  AND tpep_pickup_datetime <  '2024-02-01'
```

05 Find the peak hours.

```
SELECT hour(tpep_pickup_datetime) AS hr, COUNT(*) AS trips
FROM 'yellow_tripdata_2024-01.parquet'
-- add the WHERE filter
GROUP BY hr ORDER BY trips DESC LIMIT 5;
```

06 Find the busiest pickup zones.

Join the trips to the zone names.

```
SELECT z.Zone, COUNT(*) AS trips
FROM 'yellow_tripdata_2024-01.parquet' t
JOIN 'taxi_zone_lookup.csv' z ON t.PULocationID = z.LocationID
GROUP BY z.Zone ORDER BY trips DESC LIMIT 10;
```

07 Average tip on card payments.

Tips are recorded for card payments (payment_type 1), so measure those.

```
SELECT ROUND(AVG(tip_amount / fare_amount) * 100, 1) AS tip_pct
FROM 'yellow_tripdata_2024-01.parquet'
WHERE payment_type = 1 AND fare_amount > 0;
```

08 Chart it.

```
import matplotlib.pyplot as plt
df = con.sql("""SELECT hour(tpep_pickup_datetime) AS hr,
  COUNT(*) AS trips FROM 'yellow_tripdata_2024-01.parquet'
  GROUP BY hr ORDER BY hr""").df()
df.plot(x="hr", y="trips", kind="bar", legend=False)
plt.savefig("peak_hours.png", dpi=150)
```

09 Publish three findings.

README.md on GitHub: the question, the query, the chart and one sentence per finding.

CHECK IT WORKS

Run or check	You should see
The COUNT(*) query	a few million rows for one month
Peak hours query	late afternoon and evening hours at the top
peak_hours.png	24 bars, one per hour

IF IT BREAKS

Problem	Fix
File not found	Run Python from the folder that holds the files, or use the full path.
Pickups in 2002 or 2009	Bad timestamps exist in raw data. That is why the month filter is there.
Tip % looks huge	Tiny fares blow up the ratio. Add AND fare_amount >= 3.

GO FURTHER

Download three months and compare weekday against weekend demand by zone.

INTERVIEW QUESTION YOU CAN NOW ANSWER

Why is Parquet better than CSV for analytics?

Parquet is columnar and compressed and stores each column's type. An engine reads only the columns a query needs, so files are smaller and queries are much faster.

RESUME LINE

Analysed NYC taxi trip data in Parquet with DuckDB SQL and pandas, found peak hours, top pickup zones and tipping patterns, and published the findings with charts on GitHub.

PROJECT 04 / KAFKA / FROM THE VIDEO

Real-Time Click Stream

You can run a streaming platform, produce and consume events, and recover from a crash without losing data.

TIME 3 hours	LEVEL Intermediate

TOOLS

Apache Kafka official image, KRaft mode, no ZooKeeper

confluent-kafka (Python) producer and consumer client

PostgreSQL 16 reuse the container from project 1

WHAT YOU BUILD

Piece	What it is	Detail
broker	One Kafka broker in Docker	localhost:9092
clicks	Topic	3 partitions, replication 1
producer.py	Sends fake click events	about 10 a second, keyed by user
consumer.py	Consumer group 'counter'	clicks per page per minute
page_minute	Postgres table	primary key (minute, page)

BUILD IT

01 Start Kafka.

The official image starts a single broker in KRaft mode.

```
docker run -d --name broker -p 9092:9092 apache/kafka:latest
```

02 Create the topic.

```
docker exec -it broker /opt/kafka/bin/kafka-topics.sh \
  --bootstrap-server localhost:9092 --create \
  --topic clicks --partitions 3 --replication-factor 1
```

03 Create the results table.

In psql on the Postgres container from project 1.

```
CREATE TABLE page_minute (
  minute timestamptz, page text, clicks int,
  PRIMARY KEY (minute, page));
```

04 Write the producer.

```
pip install confluent-kafka "psycopg[binary]"

# producer.py
import json, random, time
from confluent_kafka import Producer
p = Producer({"bootstrap.servers": "localhost:9092"})
pages = ["/home", "/pricing", "/blog", "/signup"]
while True:
    e = {"page": random.choice(pages),
         "user": random.randint(1, 500), "ts": time.time()}
    p.produce("clicks", key=str(e["user"]), value=json.dumps(e))
    p.poll(0)
    time.sleep(0.1)
```

05 Write the consumer.

Auto-commit is off. Offsets are committed only after the counts are written.

```
# consumer.py
import json, psycopg
from collections import Counter
from datetime import datetime, timezone
from confluent_kafka import Consumer
c = Consumer({"bootstrap.servers": "localhost:9092",
  "group.id": "counter", "auto.offset.reset": "earliest",
  "enable.auto.commit": False})
c.subscribe(["clicks"])
db = psycopg.connect("postgresql://postgres:pass@localhost:5432/postgres")
counts, n = Counter(), 0
while True:
    m = c.poll(1.0)
    if m is None or m.error(): continue
    e = json.loads(m.value())
    minute = datetime.fromtimestamp(e["ts"], timezone.utc).replace(second=0, microsecond=0)
    counts[(minute, e["page"])] += 1; n += 1
    if n % 100 == 0:
        with db.transaction():
            for (mi, pg), k in counts.items():
                db.execute("""INSERT INTO page_minute VALUES (%s,%s,%s)
                  ON CONFLICT (minute, page) DO UPDATE
                  SET clicks = page_minute.clicks + EXCLUDED.clicks""", (mi, pg, k))
        c.commit(asynchronous=False); counts.clear()
```

06 Run both.

Two terminals: python producer.py and python consumer.py.

07 Crash it on purpose.

Stop the consumer with Ctrl+C, wait 30 seconds, start it again. It picks up from its last committed offset.

```
docker exec -it broker /opt/kafka/bin/kafka-consumer-groups.sh \
  --bootstrap-server localhost:9092 --describe --group counter
```

CHECK IT WORKS

Run or check	You should see
`kafka-console-consumer.sh --topic clicks --from-beginning`	a stream of JSON click events
`kafka-consumer-groups.sh --describe`	3 partitions, LAG falling back near 0 after a restart
`SELECT * FROM page_minute ORDER BY minute DESC;`	a row per page per minute

IF IT BREAKS

Problem	Fix
Consumer prints nothing	The group is already at the end. Keep the producer running, or use a new group.id.
Connection errors to 9092	docker ps to check the broker is up; the port mapping must be -p 9092:9092.
Counts jump after a crash	Messages since the last commit are replayed (at-least-once). Commit more often, or see Go further.

GO FURTHER

Store the offsets in the same Postgres transaction as the counts, so a replay can never double count.

INTERVIEW QUESTION YOU CAN NOW ANSWER

What is a consumer offset, and why commit it after processing?

The offset is the consumer's position in each partition. Committing after the write means a crash replays messages instead of losing them, which is at-least-once delivery; the write then needs to cope with replays.

RESUME LINE

Built a real-time event pipeline on Apache Kafka in Docker: a Python producer streams click events and a consumer group aggregates clicks per page per minute into PostgreSQL with manual offset commits.

PROJECT 05 / DATA MODELING / FROM THE VIDEO

Star Schema with dbt

You can turn messy raw tables into a clean star schema that analysts can query, with tests that catch bad data.

TIME 3 hours	LEVEL Intermediate

TOOLS

dbt Core + dbt-duckdb free, open source

DuckDB the warehouse, a single file

Git + GitHub version control for the models

WHAT YOU BUILD

Layer	Models	Purpose
Seeds	raw_orders, raw_customers, raw_products	CSV files you write, 20 to 50 rows each
Staging	stg_orders, stg_customers, stg_products	rename, cast, dedupe
Marts	fct_orders, dim_customers, dim_products	the star schema
Tests	schema.yml	unique, not_null, relationships

BUILD IT

01 Install dbt and start a project.

Choose duckdb when dbt init asks for a database.

```
pip install dbt-duckdb
dbt init shop
cd shop
```

02 Point the profile at a DuckDB file.

In ~/.dbt/profiles.yml.

```
shop:
  target: dev
  outputs:
    dev:
      type: duckdb
      path: shop.duckdb
```

03 Add raw data as seeds.

Write three small CSVs in seeds/. Put one duplicate order and one order for a customer that does not exist, so the tests have something to catch.

```
seeds/raw_orders.csv     order_id,customer_id,product_id,order_date,quantity,unit_price
seeds/raw_customers.csv  customer_id,first_name,country
seeds/raw_products.csv   product_id,name,category

dbt seed
```

04 Stage the orders.

models/staging/stg_orders.sql. Cast types and keep one row per order.

```
select cast(order_id as int) as order_id,
       cast(customer_id as int) as customer_id,
       cast(product_id as int) as product_id,
       cast(order_date as date) as order_date,
       quantity,
       cast(unit_price as decimal(10,2)) as unit_price
from {{ ref('raw_orders') }}
qualify row_number() over (
  partition by order_id order by order_date desc) = 1
```

05 Build the fact and dimensions.

Stage customers and products the same way, then models/marts/fct_orders.sql and dim_customers.sql:

```
-- fct_orders.sql
select order_id, order_date, customer_id, product_id,
       quantity, quantity * unit_price as revenue
from {{ ref('stg_orders') }}

-- dim_customers.sql
select customer_id, first_name, country
from {{ ref('stg_customers') }}
```

06 Add tests.

models/marts/schema.yml.

```
version: 2
models:
  - name: fct_orders
    columns:
      - name: order_id
        data_tests: [unique, not_null]
      - name: customer_id
        data_tests:
          - relationships:
              arguments:
                to: ref('dim_customers')
                field: customer_id
```

07 Build everything.

Seeds, models and tests in dependency order. The relationships test should fail on your planted bad row: fix the seed and build again.

```
dbt build
```

08 Generate the docs.

Opens a site with every model and the lineage graph. Screenshot the graph for the README.

```
dbt docs generate
dbt docs serve
```

CHECK IT WORKS

Run or check	You should see
dbt build	PASS on every model and test once the bad row is fixed
select count(*) from fct_orders	one row per order, no duplicates
dbt docs serve, lineage graph	raw seeds > staging > marts

IF IT BREAKS

Problem	Fix
Profile not found	profiles.yml must be in ~/.dbt/, or pass --profiles-dir with its folder.
Could not set lock on file	Another program has shop.duckdb open. Close it and rerun.
relationships test fails	That is the test working: an order points at a missing customer. Fix the data or filter it in staging.

GO FURTHER

Add dim_date and a monthly revenue mart, and write a singular test that fails if revenue is ever negative.

INTERVIEW QUESTION YOU CAN NOW ANSWER

What is the difference between a fact table and a dimension table?

A fact table holds measurable events, one row per order line with quantity and revenue, plus keys. Dimensions describe the who, what and when: customer, product, date. A star schema is the fact in the middle joined to its dimensions.

RESUME LINE

Modelled raw sales data into a star schema (fct_orders, dim_customers, dim_products) with dbt Core on DuckDB, with uniqueness, not-null and referential tests and generated lineage docs.

Daily Pipeline on Airflow

You can schedule a pipeline, retry failures, backfill missed days and stop bad data before it loads.

TIME 3 hours	LEVEL Intermediate

TOOLS

Apache Airflow official Docker Compose quick start

Docker Desktop give Docker at least 4 GB of memory

Your project 1 code extract, transform, load

WHAT YOU BUILD

Piece	What it is	Detail
docker-compose.yaml	Airflow's official quick-start file	scheduler, API server, Postgres, Redis, workers
dags/weather_daily.py	Your DAG	extract > transform > check > load
Schedule	@daily	with catchup to backfill
UI	localhost:8080	login airflow / airflow

BUILD IT

01 Get the compose file.

Open the Running Airflow in Docker page and copy its curl line for docker-compose.yaml into a new folder.

```
mkdir airflow && cd airflow
# paste the curl -LfO line from the Airflow docs page
```

02 Prepare folders and your user ID.

From the Airflow docs.

```
mkdir -p ./dags ./logs ./plugins ./config
echo -e "AIRFLOW_UID=$(id -u)" > .env
```

03 Start Airflow.

The init step sets up the database and the airflow login.

```
docker compose up airflow-init
docker compose up -d
```

04 Write the DAG.

dags/weather_daily.py. Airflow 3 imports from airflow.sdk (on Airflow 2 use airflow.decorators). Put your project 1 functions inside the tasks.

```
from datetime import datetime, timedelta
from airflow.sdk import dag, task

@dag(schedule="@daily", start_date=datetime(2026, 11, 1),
     catchup=True,
     default_args={"retries": 2, "retry_delay": timedelta(minutes=5)})
def weather_daily():
    @task
    def extract(): ...          # call the API, return rows
    @task
    def transform(rows): ...    # clean and type them
    @task
    def check(rows):
        if not rows or any(r["temp_max"] is None for r in rows):
            raise ValueError("bad rows, stopping before load")
        return rows
    @task
    def load(rows): ...         # upsert into Postgres
    load(check(transform(extract())))

weather_daily()
```

05 Turn it on.

In the UI, unpause weather_daily. With catchup on, Airflow runs every day since start_date: that is a backfill.

06 Alert on failure.

Add an on_failure_callback to default_args that posts to a free Slack or Discord webhook with requests.post.

```
import requests
WEBHOOK_URL = "paste your webhook URL"

def alert(context):
    requests.post(WEBHOOK_URL, json={"content":
      f"{context['task_instance'].task_id} failed"})

# then add to default_args in @dag:  "on_failure_callback": alert
```

07 Break it on purpose.

Make check raise once and watch the retries, the red task and the alert.

CHECK IT WORKS

Run or check	You should see
DAGs page	weather_daily listed with no import error banner
Grid view	a green run for each day since start_date
Graph view	extract > transform > check > load
Task logs	row counts from your load step

IF IT BREAKS

Problem	Fix
UI keeps restarting	Give Docker Desktop at least 4 GB of memory, as the Airflow docs say.
DAG not listed	The file must be in ./dags. Read the import error banner at the top of the DAGs page.
Permission errors on Linux	Set AIRFLOW_UID in .env (step 2) and restart.

GO FURTHER

Add a sensor that waits for the API to respond before extract runs.

INTERVIEW QUESTION YOU CAN NOW ANSWER

What is a DAG, and why must its tasks be idempotent?

A directed acyclic graph of tasks and their dependencies. Retries and backfills rerun tasks, so each one has to give the same result when it runs again.

RESUME LINE

Orchestrated a daily ETL pipeline in Apache Airflow (Docker Compose) with extract, transform, data-quality and load tasks, retries, backfill and failure alerts.

PROJECT 07 / DATA LAKES

Partitioned Data Lake on Your Laptop

You understand how lakes store data in folders by partition, and why a filter on the partition makes queries fast.

TIME 1.5 hours	LEVEL Beginner

TOOLS

DuckDB writes and reads partitioned Parquet

NYC TLC trip data three months of yellow taxi Parquet

WHAT YOU BUILD

Path	Holds	Note
raw/	yellow_tripdata_2024-01 to -03.parquet	files as downloaded
lake/trips/year=2024/month=1/	January trips	hive-style partition folders
lake/trips/year=2024/month=2/ ...	one folder per month	written by DuckDB

BUILD IT

01 **Download three months.**
January to March 2024 Yellow Taxi Trip Records into raw/.

02 **Write the lake.**
DuckDB adds the partition columns and writes one folder per month.

```
import duckdb, os
os.makedirs("lake", exist_ok=True)
con = duckdb.connect()
con.sql("""COPY (
  SELECT *, year(tpep_pickup_datetime) AS year,
         month(tpep_pickup_datetime) AS month
  FROM read_parquet('raw/*.parquet')
  WHERE tpep_pickup_datetime >= '2024-01-01'
    AND tpep_pickup_datetime <  '2024-04-01'
) TO 'lake/trips' (FORMAT PARQUET, PARTITION_BY (year, month))""")
```

03 **Query one month.**
With hive_partitioning on, the month filter only opens that folder.

```
con.sql("""SELECT COUNT(*) FROM read_parquet(
  'lake/trips/*/*/*.parquet', hive_partitioning = true)
  WHERE month = 2""").show()
```

04 **Prove the pruning.**
Compare the plans. The filtered one reads fewer files.

```
con.sql("EXPLAIN ANALYZE SELECT COUNT(*) FROM read_parquet("
  "'lake/trips/*/*/*.parquet', hive_partitioning=true) WHERE month = 2").show()
```

CHECK IT WORKS

Run or check	You should see
ls lake/trips/year=2024	month=1, month=2, month=3
The EXPLAIN ANALYZE output	Scanning Files: 1/3
Query with WHERE month = 2	February's count only, returned quickly

IF IT BREAKS

Problem	Fix
Binder error on month	Add hive_partitioning = true so DuckDB reads the folder names as columns.
A month=12 folder appears	Bad timestamps in raw data. Keep the date filter in step 2.

GO FURTHER

Add a daily partition and compare file counts and query times.

INTERVIEW QUESTION YOU CAN NOW ANSWER

What is hive-style partitioning?

Data is stored in folders named column=value, such as month=2. Engines read the folder names as columns and skip folders a filter rules out.

RESUME LINE

Built a hive-partitioned Parquet data lake from NYC taxi data with DuckDB and demonstrated partition pruning with query plans.

PROJECT 08 / DATA PIPELINES

Incremental Load with a Watermark

You can load only what changed since the last run, which is how real pipelines handle big tables.

TIME 2 hours	LEVEL Intermediate

TOOLS

PostgreSQL 16 the container from project 1

Python + psycopg runs the load

WHAT YOU BUILD

Table	Role	Key columns
src_orders	Source system	order_id, amount, updated_at
dw_orders	Warehouse copy	same columns, primary key order_id
etl_state	Remembers the last load	pipeline, watermark

BUILD IT

01 Create the tables.

```
CREATE TABLE src_orders (order_id int PRIMARY KEY, amount numeric,
  updated_at timestamptz DEFAULT now());
CREATE TABLE dw_orders (LIKE src_orders INCLUDING ALL);
CREATE TABLE etl_state (pipeline text PRIMARY KEY, watermark timestamptz);
INSERT INTO etl_state VALUES ('orders', '1970-01-01');
```

02 Add source rows.

```
INSERT INTO src_orders (order_id, amount)
SELECT g, round((random() * 100)::numeric, 2)
FROM generate_series(1, 1000) g;
```

03 Write the incremental load.

One transaction: copy rows newer than the watermark, then move the watermark.

```
BEGIN;
INSERT INTO dw_orders
SELECT * FROM src_orders
WHERE updated_at > (SELECT watermark FROM etl_state WHERE pipeline = 'orders')
ON CONFLICT (order_id) DO UPDATE
  SET amount = EXCLUDED.amount, updated_at = EXCLUDED.updated_at;
UPDATE etl_state SET watermark =
  (SELECT max(updated_at) FROM dw_orders) WHERE pipeline = 'orders';
COMMIT;
```

04 Change some rows and load again.

Only the changed rows move on the second run.

```
UPDATE src_orders SET amount = amount + 1, updated_at = now()
WHERE order_id <= 10;
-- rerun step 3
```

CHECK IT WORKS

Run or check	You should see
SELECT COUNT(*) FROM dw_orders;	1000
Rows inserted or updated on the second run	10
SELECT * FROM etl_state;	the watermark moved forward

IF IT BREAKS

Problem	Fix
Every run reloads everything	The watermark is not being updated. Check the UPDATE ran inside the transaction.
Changes are missed	The source must set updated_at on every change. Add a trigger if it does not.

GO FURTHER

Wrap step 3 in Python and log rows moved per run to a table.

INTERVIEW QUESTION YOU CAN NOW ANSWER

Full load or incremental load: when would you use each?

Full loads are simple and fine for small tables. Incremental loads move only new or changed rows using a watermark such as updated_at, which is needed once tables are large; the risk is missing changes that do not touch the watermark column.

RESUME LINE

Implemented watermark-based incremental loading in PostgreSQL, moving only changed rows per run with transactional state tracking.

PROJECT 09 / DATA MODELING

Customer History with dbt Snapshots

You can keep history when a record changes, the slowly changing dimension type 2 pattern asked about in interviews.

TIME 1 hour	LEVEL Intermediate

TOOLS

dbt Core + dbt-duckdb builds on project 5

WHAT YOU BUILD

Column	Added by dbt	Meaning
dbt_valid_from	yes	when this version became true
dbt_valid_to	yes	when it stopped; empty for the current row
dbt_scd_id	yes	a unique ID for each version

BUILD IT

01 **Add a snapshot.**

snapshots/customers_snapshot.sql in your project 5 folder.

```
{% snapshot customers_snapshot %}
{{ config(target_schema='snapshots', unique_key='customer_id',
          strategy='check', check_cols=['country']) }}
select * from {{ ref('stg_customers') }}
{% endsnapshot %}
```

02 **Take the first snapshot.**

```
dbt snapshot
```

03 **Change a customer.**

Edit one customer's country in seeds/raw_customers.csv, then reload and snapshot again.

```
dbt seed
dbt run --select stg_customers
dbt snapshot
```

04 **Read the history.**

```
select customer_id, country, dbt_valid_from, dbt_valid_to
from snapshots.customers_snapshot
where customer_id = 1 order by dbt_valid_from;
```

CHECK IT WORKS

Run or check	You should see
The history query	two rows: the old country closed, the new one open

IF IT BREAKS

Problem	Fix
No second row	The changed column must be in check_cols, and stg_customers must be rebuilt first.

GO FURTHER

Join fct_orders to the snapshot on order_date between valid_from and valid_to, so each order shows the country at the time.

INTERVIEW QUESTION YOU CAN NOW ANSWER

What is a type 2 slowly changing dimension?

When an attribute changes, the old row is closed with an end date and a new row is added, so history is kept and facts can be joined to the version that was true at the time.

RESUME LINE

Implemented SCD type 2 history for a customer dimension using dbt snapshots, enabling point-in-time reporting.

PROJECT 10 / DATA QUALITY

Data Quality Report in SQL

You can measure data quality and stop a pipeline when the numbers are wrong, before anyone builds a report on them.

TIME 1.5 hours	LEVEL Beginner

TOOLS

DuckDB + Python runs the checks

NYC TLC trip data January 2024 from project 3

WHAT YOU BUILD

Check	Rule	Fails when
null_fare	fare_amount is not null	more than 0.1% nulls
negative_total	total_amount >= 0	any negative rows
out_of_month	pickup inside January 2024	more than 0.1% outside
zero_distance	trip_distance > 0	more than 2% zero

BUILD IT

01 Write each check as one SQL query.

Each returns the share of bad rows.

```
CHECKS = {
 "null_fare": "avg(CASE WHEN fare_amount IS NULL THEN 1 ELSE 0 END)",
 "negative_total": "avg(CASE WHEN total_amount < 0 THEN 1 ELSE 0 END)",
 "out_of_month": "avg(CASE WHEN tpep_pickup_datetime < '2024-01-01'"
    " OR tpep_pickup_datetime >= '2024-02-01' THEN 1 ELSE 0 END)",
 "zero_distance": "avg(CASE WHEN trip_distance = 0 THEN 1 ELSE 0 END)"}
LIMITS = {"null_fare": 0.001, "negative_total": 0.0,
          "out_of_month": 0.001, "zero_distance": 0.02}
```

02 Run them and fail on a breach.

Exit code 1 is what stops a scheduler or a CI job.

```
import duckdb, sys
T = "'yellow_tripdata_2024-01.parquet'"
bad = []
for name, expr in CHECKS.items():
    v = duckdb.sql(f"SELECT {expr} FROM {T}").fetchall()[0][0]
    status = "FAIL" if v > LIMITS[name] else "ok"
    print(f"{name:15} {v:.4%}  {status}")
    if status == "FAIL": bad.append(name)
sys.exit(1 if bad else 0)
```

03 Save the results.

Append each run's numbers to a checks table or CSV so you can chart quality over time.

CHECK IT WORKS

Run or check	You should see
python checks.py	one line per check with a percentage
echo $? after a failing run	1

IF IT BREAKS

Problem	Fix
Every check fails	Real raw data is messy. Read the numbers, then set limits you can defend in an interview.

GO FURTHER

Run checks.py in the Airflow DAG from project 6 as the check task.

INTERVIEW QUESTION YOU CAN NOW ANSWER

How do you know a pipeline's data is right?

Automated checks on every run: nulls, ranges, duplicates, freshness and row counts, with thresholds that fail the run before bad data reaches reports.

RESUME LINE

Built automated data-quality checks in SQL (nulls, ranges, freshness) that fail the pipeline on threshold breaches.

PROJECT 11 / BIG DATA

Spark Batch Job

You can use Apache Spark, the engine most big data jobs ask for, and read its UI.

TIME 2 hours	LEVEL Intermediate

TOOLS

PySpark pip install pyspark; needs Java 17 or newer

NYC TLC trip data the three months from project 7

WHAT YOU BUILD

Stage	What happens	Output
Read	Three months of Parquet	one DataFrame
Transform	Clean, add pickup date and hour	trips
Aggregate	Trips and revenue per day and zone	daily_zone
Write	Parquet partitioned by month	out/daily_zone/

BUILD IT

01 Install.

Check Java first with java -version.

```
pip install pyspark
```

02 Start a local session and read.

```
from pyspark.sql import SparkSession, functions as F
spark = SparkSession.builder.master("local[*]").appName("taxi").getOrCreate()
df = spark.read.parquet("raw/yellow_tripdata_2024-0*.parquet")
```

03 Clean and aggregate.

```
daily = (df.filter((F.col("trip_distance") > 0) & (F.col("total_amount") > 0))
  .withColumn("day", F.to_date("tpep_pickup_datetime"))
  .withColumn("month", F.month("tpep_pickup_datetime"))
  .groupBy("month", "day", "PULocationID")
  .agg(F.count("*").alias("trips"),
       F.round(F.sum("total_amount"), 2).alias("revenue")))
```

04 Write it out.

```
daily.write.mode("overwrite").partitionBy("month").parquet("out/daily_zone")
```

05 Read the Spark UI.

While the job runs, open localhost:4040. Find the shuffle in the stage list and screenshot it.

CHECK IT WORKS

Run or check	You should see
ls out/daily_zone	month=1, month=2, month=3 folders
localhost:4040 > Stages	a stage with shuffle read and write

IF IT BREAKS

Problem	Fix
Java gateway process exited	Install Java 17 or newer and set JAVA_HOME.
Out of memory	Read one month first, or start with .config("spark.driver.memory", "4g").

GO FURTHER

Rewrite the same job in DuckDB and compare the run time on your laptop.

INTERVIEW QUESTION YOU CAN NOW ANSWER

What is a shuffle in Spark?

Moving data between partitions so rows with the same key end up together, needed for groupBy and joins. It is the most expensive part of most jobs.

RESUME LINE

Wrote a PySpark batch job that cleans and aggregates three months of NYC taxi trips and writes partitioned Parquet output.

PROJECT 12 / AUTOMATED ORCHESTRATION

Scheduled Pipeline on GitHub Actions

You can run a pipeline on a schedule in the cloud for free, with no server to look after.

TIME 1 hour	LEVEL Beginner

TOOLS

GitHub Actions free for public repositories

Open-Meteo API the source from project 1

WHAT YOU BUILD

File	What it does	Detail
fetch.py	Pulls today's forecast, appends to a CSV	data/weather.csv
.github/workflows/daily.yml	Runs fetch.py every morning	cron 0 6 * * * (UTC)

BUILD IT

01 Write fetch.py.

Reuse the extract and transform code from project 1, then append the rows to data/weather.csv with df.to_csv(..., mode="a").

02 Add the workflow.

.github/workflows/daily.yml in a public repo.

```
name: daily-weather
on:
  schedule:
    - cron: "0 6 * * *"
  workflow_dispatch:
permissions:
  contents: write
jobs:
  run:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-python@v5
        with:
          python-version: "3.12"
      - run: pip install requests pandas
      - run: python fetch.py
      - run: |
          git config user.name "github-actions"
          git config user.email "github-actions@users.noreply.github.com"
          git add data/
          git commit -m "daily data" || echo "no changes"
          git push
```

03 Run it once by hand.

Actions tab > daily-weather > Run workflow.

CHECK IT WORKS

Run or check	You should see
Actions tab	a green run
data/weather.csv	new rows each day, committed by github-actions

IF IT BREAKS

Problem	Fix
Push rejected (403)	The permissions: contents: write block is missing.
Schedule stopped running	GitHub pauses schedules in inactive public repos. The daily commit keeps it active; re-enable it in the Actions tab if needed.

GO FURTHER

Add the checks from project 10 as a step so a bad day fails the run.

INTERVIEW QUESTION YOU CAN NOW ANSWER

How would you schedule a small pipeline without running a server?

A scheduled CI job such as GitHub Actions with a cron trigger: it runs the script on a hosted runner, stores the output, and shows every run's log.

RESUME LINE

Automated a daily data pipeline on GitHub Actions with a cron schedule, committing new data to the repository on each run.

PROJECT 13 / DATA SERVING

Data API with FastAPI

You can serve warehouse data to other apps through an API, not just to analysts.

TIME 1.5 hours	LEVEL Intermediate

TOOLS

FastAPI + Uvicorn free, open source

DuckDB queries the taxi files from project 3

WHAT YOU BUILD

Endpoint	Returns	Example
GET /zones/top	Top 10 pickup zones	/zones/top
GET /zones/{zone_id}/hours	Trips per hour for one zone	/zones/132/hours
GET /docs	Interactive docs	built in

BUILD IT

01 **Install.**

```
pip install fastapi uvicorn duckdb
```

02 **Write main.py.**

Parameters are passed with ?, never pasted into the SQL string.

```
import duckdb
from fastapi import FastAPI
app = FastAPI()
T = "'yellow_tripdata_2024-01.parquet'"

@app.get("/zones/top")
def top():
    q = f"""SELECT PULocationID AS zone, COUNT(*) AS trips FROM {T}
            GROUP BY 1 ORDER BY 2 DESC LIMIT 10"""
    return duckdb.sql(q).df().to_dict("records")

@app.get("/zones/{zone_id}/hours")
def hours(zone_id: int):
    q = f"""SELECT hour(tpep_pickup_datetime) AS hr, COUNT(*) AS trips
            FROM {T} WHERE PULocationID = ? GROUP BY 1 ORDER BY 1"""
    return duckdb.execute(q, [zone_id]).df().to_dict("records")
```

03 **Run it.**

Then open localhost:8000/docs and try both endpoints.

```
uvicorn main:app --reload
```

CHECK IT WORKS

Run or check	You should see
localhost:8000/zones/top	JSON list of 10 zones
localhost:8000/docs	both endpoints listed with a Try it out button

IF IT BREAKS

Problem	Fix
422 error	zone_id must be a number.
File not found	Start uvicorn from the folder with the Parquet file.

GO FURTHER

Precompute the results into a table in project 2's warehouse and cache responses for a minute.

INTERVIEW QUESTION YOU CAN NOW ANSWER

Why use query parameters instead of building SQL strings?

Parameters are sent separately from the SQL, so user input cannot change the query. That prevents SQL injection.

RESUME LINE

Built a REST API with FastAPI and DuckDB that serves aggregated taxi trip data, with parameterised queries and auto-generated docs.

PROJECT 14 / DATA SERVING

Self-Hosted Dashboard with Metabase

You can put a business dashboard on top of your own database, so non-technical people can use the data.

TIME 1 hour	LEVEL Beginner

TOOLS

Metabase (open source) free, runs in Docker

PostgreSQL the weather data from project 1

WHAT YOU BUILD

Piece	What it is	Where
Metabase	Dashboard tool	localhost:3000
Database	Postgres from project 1	host.docker.internal:5432 on Mac and Windows

BUILD IT

01 Start Metabase.

```
docker run -d -p 3000:3000 --name metabase metabase/metabase
```

02 Connect the database.

Open localhost:3000, create the admin login, add a PostgreSQL database. Host host.docker.internal, port 5432, database postgres, user postgres, password pass. On Linux use your machine's IP address.

03 Build three questions.

Max temperature by day (line), total rain by city (bar), and the latest reading per city (table).

04 Make the dashboard.

New dashboard, add the three questions, add a date filter wired to all three. Screenshot it for the README.

CHECK IT WORKS

Run or check	You should see
localhost:3000	the dashboard with three charts and a working date filter

IF IT BREAKS

Problem	Fix
Can't connect to the database	Inside Docker, localhost is the container itself. Use host.docker.internal or your IP.
Page loads slowly at first	Metabase takes a minute or two to start. docker logs metabase shows progress.

GO FURTHER

Point Metabase at the dbt marts from project 5 instead of raw tables.

INTERVIEW QUESTION YOU CAN NOW ANSWER

Why do analysts query marts instead of raw tables?

Marts are cleaned, tested and named for the business, so every dashboard uses the same definitions and numbers agree.

RESUME LINE

Deployed Metabase in Docker on PostgreSQL and built a filtered weather dashboard for non-technical users.

PROJECT 15 / DATA PIPELINES

Web Log Pipeline

You can turn messy text logs into clean tables and answer operations questions from them.

TIME 2 hours	LEVEL Intermediate

TOOLS

Python + pandas + pyarrow re for parsing, pyarrow for Parquet

DuckDB queries the result

WHAT YOU BUILD

Stage	Input	Output
Generate	make_logs.py	access.log, 100,000 lines
Parse	access.log	logs.parquet
Report	logs.parquet	top paths, error rate per hour

BUILD IT

01 Install.

pandas needs pyarrow to write Parquet.

```
pip install pandas pyarrow duckdb
```

02 Make sample logs.

A fake web server log in the common format.

```
import random, datetime as dt
paths = ["/", "/login", "/api/items", "/cart", "/checkout"]
t = dt.datetime(2026, 11, 1)
with open("access.log", "w") as f:
    for i in range(100_000):
        t += dt.timedelta(seconds=random.randint(0, 3))
        code = random.choices([200, 404, 500], [95, 3, 2])[0]
        ip = f"10.0.{random.randint(0,255)}.{random.randint(1,254)}"
        f.write(f'{ip} - - [{t:%d/%b/%Y:%H:%M:%S} +0000] '
                f'"GET {random.choice(paths)} HTTP/1.1" {code} '
                f'{random.randint(200, 5000)}\n')
```

03 Parse with a regex.

```
import re, pandas as pd
pat = re.compile(r'(\S+) - - \[(.+?)\] "(\S+) (\S+) \S+" (\d{3}) (\d+)')
rows = [pat.match(l).groups() for l in open("access.log")]
df = pd.DataFrame(rows, columns=["ip","ts","method","path","status","bytes"])
df["ts"] = pd.to_datetime(df["ts"], format="%d/%b/%Y:%H:%M:%S %z")
df[["status","bytes"]] = df[["status","bytes"]].astype(int)
df.to_parquet("logs.parquet")
```

04 Report.

```
SELECT date_trunc('hour', ts) AS hr,
  ROUND(100.0 * avg(CASE WHEN status >= 500 THEN 1 ELSE 0 END), 2) AS pct_5xx
FROM 'logs.parquet' GROUP BY hr ORDER BY hr;
```

CHECK IT WORKS

Run or check	You should see
SELECT COUNT(*) FROM 'logs.parquet';	100000
The 5xx query	about 2% per hour, matching the generator

IF IT BREAKS

Problem	Fix
AttributeError: 'NoneType'	A line did not match the regex. Print the first failing line and adjust the pattern.

GO FURTHER

Stream the log lines into Kafka (project 4) and compute the error rate per minute live.

INTERVIEW QUESTION YOU CAN NOW ANSWER

How would you turn unstructured logs into something you can query?

Parse each line into fields with a pattern, give the fields types, store them in a columnar format like Parquet, then query with SQL.

RESUME LINE

Built a log-processing pipeline that parses web server logs with Python regex into Parquet and reports traffic and error rates with SQL.

PROJECT 16 / CAPSTONE

End-to-End Capstone

You can design and run a whole data platform, which is the project interviewers ask you to walk through.

TIME 1 to 2 weekends	LEVEL Advanced

TOOLS

Airflow project 6

PostgreSQL project 1

dbt Core project 5, with the dbt-postgres adapter

Metabase project 14

WHAT YOU BUILD

Layer	Tool	What it does
Ingest	Airflow task	pulls the API daily into raw tables
Store	Postgres	raw schema and analytics schema
Transform	dbt	staging and marts, with tests
Check	Airflow task	fails the run if dbt tests fail
Serve	Metabase	dashboard on the marts

BUILD IT

01 Draw the architecture first.
One diagram: source > Airflow > Postgres raw > dbt > marts > Metabase. It goes at the top of the README.

02 Ingest.
Reuse project 6's DAG, loading into a raw schema in Postgres.

03 Transform with dbt on Postgres.
Install the Postgres adapter and point a profile at the same database.

```
pip install dbt-postgres
# profiles.yml: type: postgres, host, user, password, dbname, schema: analytics
```

04 Run dbt from Airflow.
Add a final task after load. If a dbt test fails, the task fails and you get the alert from project 6.

```
@task.bash
def dbt_build():
    return "cd /opt/airflow/dbt && dbt build"
```

05 Serve it.
Connect Metabase to the analytics schema and build the dashboard.

06 Write it up.
README: the diagram, why each tool, how to run it with one command, and what broke along the way.

CHECK IT WORKS

Run or check	You should see
Airflow grid	ingest > dbt_build green each day
Metabase	dashboard updates after each run

IF IT BREAKS

Problem	Fix
dbt not found inside Airflow	Extend the Airflow image with dbt installed, as the Airflow docs describe for adding Python packages.

GO FURTHER

Swap Postgres for the BigQuery warehouse from project 2.

INTERVIEW QUESTION YOU CAN NOW ANSWER

Walk me through a pipeline you built.

Use this project: the source, how it is ingested and scheduled, where raw data lands, how it is modelled and tested, how failures alert, and who uses the output.

RESUME LINE

Designed and built an end-to-end data platform: Airflow ingestion, PostgreSQL storage, dbt transformations with tests and a Metabase dashboard.

WHAT TO DO NEXT

A 4-week plan

Week	Build	Why
1	Projects 1 and 3	Load data, then query it
2	Projects 2 and 5	A cloud warehouse and a star schema
3	Projects 6 and 10	Schedule it and check the data
4	Project 4	Your first streaming pipeline
After	Pick from 7 to 16	Match the job posts you are applying to

PAIR THE PROJECTS WITH A CERT

Microsoft DP-900: Azure Data Fundamentals

Projects prove you can do the work. A cert gets your resume past the filter. DP-900 covers the core ideas behind these projects: relational and non-relational data, and analytics workloads like data warehouses. Practise the questions while you build.

Try 15 free DP-900 questions. No sign-up needed.
getuprep.com/microsoft/dp-900

Built one? Post it on LinkedIn and GitHub with the resume line, and tag @getuacademy.

If this helped, please consider following @getuacademy.