TL;DR Quick Start

- **Set Up Cursor** – Install, link repo, add Python/SQL/PySpark/dbt support. **Example:** `cursor link`
- **Organize Repo** – Separate ingestion, transforms, orchestration; add README and consistent names. **Example:** `/ingestion/api_to_s3.py`
- **Generate Pipelines** – Use Cursor prompts to create scripts automatically. **Example:** `"Generate Python ingestion from REST API to S3 with logging, retries, and pagination."`
- **Automate Data Quality** – Run checks for nulls, duplicates, schema drift. **Example:** `def check_nulls(df): return df.isnull().sum()[df.isnull().sum() > 0]`
- **Debug & Optimize** – Trace errors, optimize PySpark, rewrite SQL/dbt. **Example Prompt:** `"Optimize this PySpark job with broadcast joins and caching."`

---

1. Setting Up Cursor for Data Engineering Projects

1.1 Organize Repo for AI-Friendly Context

- Separate scripts by function:

```
/ingestion
/transforms
/orchestration
```

- Include metadata files: `README.md`, `schemas.yaml`, `data_catalog.md`.
- Maintain consistent naming conventions.
- Document schema relationships in a single source of truth.

1.2 Configure Cursor With Your Tech Stack

1. Install Cursor:
2. Python: `pip install cursor-ai`
3. Node.js: `npm install cursor-ai`
4. Add language kernels: SQL, PySpark, YAML.
5. Link version control (Git).
6. Connect optional data sources for realistic prompts.

1.3 Recommended Cursor Settings

- **Context depth:** Full repo coverage.
- **Auto-suggestion mode:** On.

- **Prompt history:** Enable.
- **Security filters:** Exclude sensitive credentials.
- **Custom snippets:** Preload common templates.

---

2. Generating Ingestion Pipelines

2.1 Starter Prompt for Ingestion

```
Rewrite this ingestion script to support dynamic pagination, exponential
backoff,
and automatic retries for 429/500 errors. Add inline comments explaining each
step.
Save the improved version to /ingestion/api_to_s3_v2.py.
```

2.2 Example Python Ingestion Script

```
import time, json, requests, boto3
s3 = boto3.client("s3")

def fetch_page(url, retries=5):
    for attempt in range(retries):
        response = requests.get(url)
        if response.status_code in [429, 500]:
            time.sleep(2 ** attempt)
            continue
        response.raise_for_status()
        return response.json()

def ingest_to_s3(base_url, bucket, key_prefix):
    page = 1
    has_more = True
    while has_more:
        url = f"{base_url}?page={page}"
        data = fetch_page(url)
        s3.put_object(Bucket=bucket, Key=f"{key_prefix}/page_{page}.json",
Body=json.dumps(data))
        has_more = data.get("has_more", False)
        page += 1
```

2.3 Ingestion Checklist

- Validate input schema.
- Handle API rate limits.
- Include logging.

- Save raw snapshots.
- Support incremental and full-load modes.

---

3. Handling Schema Changes

- Detect new/missing fields automatically.
- Update parsing logic and transformations.
- Notify downstream transformations on changes.

---

4. Transformations

4.1 SQL Model Generation

Starter Prompt:

```
Generate a clean SQL transformation model that removes duplicates, applies
column-level type casting,
and includes a validated SELECT block. Output to /transforms/sql/
clean_orders.sql.
```

Example SQL Output:

```
WITH deduped AS (
    SELECT *, ROW_NUMBER() OVER (PARTITION BY order_id ORDER BY updated_at DESC)
AS rn
    FROM raw.orders
),
typed AS (
    SELECT order_id, CAST(user_id AS BIGINT) AS user_id,
           CAST(created_at AS TIMESTAMP) AS created_at,
           amount::DECIMAL(10,2) AS amount
    FROM deduped
    WHERE rn = 1
)
SELECT * FROM typed WHERE amount > 0;
```

4.2 PySpark Optimization

Prompt:

```
Optimize this PySpark job: switch to DataFrame APIs, reduce shuffle, use
broadcast joins,
and cache where needed. Save as /transforms/pyspark/orders_v2.py.
```

Example PySpark Script:

```
from pyspark.sql import SparkSession, functions as F
from pyspark.sql.functions import broadcast

spark = SparkSession.builder.getOrCreate()
orders = spark.read.parquet("s3://lake/orders")
users = spark.read.parquet("s3://lake/users")
users_b = broadcast(users)

orders_clean = (
    orders.filter(F.col("amount") > 0)
    .join(users_b, "user_id", "left")
    .withColumn("created_at", F.to_timestamp("created_at"))
)
orders_clean.cache()
orders_clean.write.mode("overwrite").parquet("s3://warehouse/orders_clean")
```

4.3 dbt Workflow

Prompt:

```
Convert this SQL into a dbt model with schema tests, documented sources,
and incremental logic using updated_at. Output to models/staging/
stg_payments.sql.
```

Example dbt Model:

```
{{ config(
    materialized='incremental',
    unique_key='payment_id',
    incremental_strategy='delete+insert'
) }}

WITH source AS (
    SELECT * FROM {{ source('billing', 'payments') }}
),
clean AS (
```

```
    SELECT payment_id, user_id, amount::decimal(10,2) AS amount,
           CAST(updated_at AS timestamp) AS updated_at
    FROM source
)
SELECT * FROM clean
{% if is_incremental() %}
WHERE updated_at > (SELECT MAX(updated_at) FROM {{ this }})
{% endif %}
```

---

5. Automating Data Quality Checks

5.1 Quality Check Prompts

```
Create a data quality module validating nulls, schema drift, duplicates,
and out-of-range values. Save to /quality/checks.py with clear pass/fail
messages.
```

5.2 Example Python Checker

```
def check_nulls(df):
    return df.isnull().sum()[df.isnull().sum() > 0]

def check_duplicates(df, key_cols):
    return df.duplicated(subset=key_cols).sum()

def check_range(df, col, min_val, max_val):
    return len(df[(df[col] < min_val) | (df[col] > max_val)])

def run_quality_checks(df):
    print("Null Check:", check_nulls(df))
    print("Duplicate Check:", check_duplicates(df, ["id"]))
    print("Range Check Amount:", check_range(df, "amount", 0, 5000))
```

5.3 CI/CD Integration

- Run checks in pre-commit or CI/CD stage.
- Fail builds if issues are detected.
- Store reports for auditing.

---

6. Debugging & Refactoring Pipelines

6.1 Debugging Workflow

1. Identify failing module.
2. Prompt Cursor:

```
Trace the data flow to this module; highlight schema/type mismatches.
```

3. Apply suggested fixes.
4. Repeat until resolved.

6.2 Example Debug Fix

```
# Fix for missing 'created_at' due to join
fixed = (
    orders
    .join(events.withColumnRenamed("created_at", "event_created_at"),
"order_id")
    .withColumn("created_at", F.coalesce("event_created_at",
"orders.created_at"))
)
```

6.3 Refactoring Patterns

- Modularization: reusable functions.
- Parameterization: config variables.
- Standardized logging.
- Reusable SQL, PySpark, or dbt templates.

---

7. Creating Reusable Runbooks

7.1 Runbook Structure Template

- **Overview:** Purpose, inputs, outputs.
- **Dependencies:** Tables, APIs, upstream transformations.
- **Prompt library:** Cursor prompts for scripts and checks.
- **Execution steps:** Commands, tests, CI/CD integration.
- **Validation & QA:** Expected outputs, automated checks.

7.2 Example: API → S3 Ingestion Runbook

- **Overview:** Ingest customer events daily.
- **Dependencies:** `/schemas/customer_events.yaml`, API token.

- **Prompts:** Cursor-generated Python ingestion, null checks, retry logic.
- **Execution:** `python ingestion/api_to_s3.py`
- **QA:** Run automated null checks and save report to `/reports/`.

7.3 Storing Runbooks

- Save as `.cursor` or Markdown in `/runbooks/`.
- Version control runbooks.
- Link scripts/prompts for instant execution.
