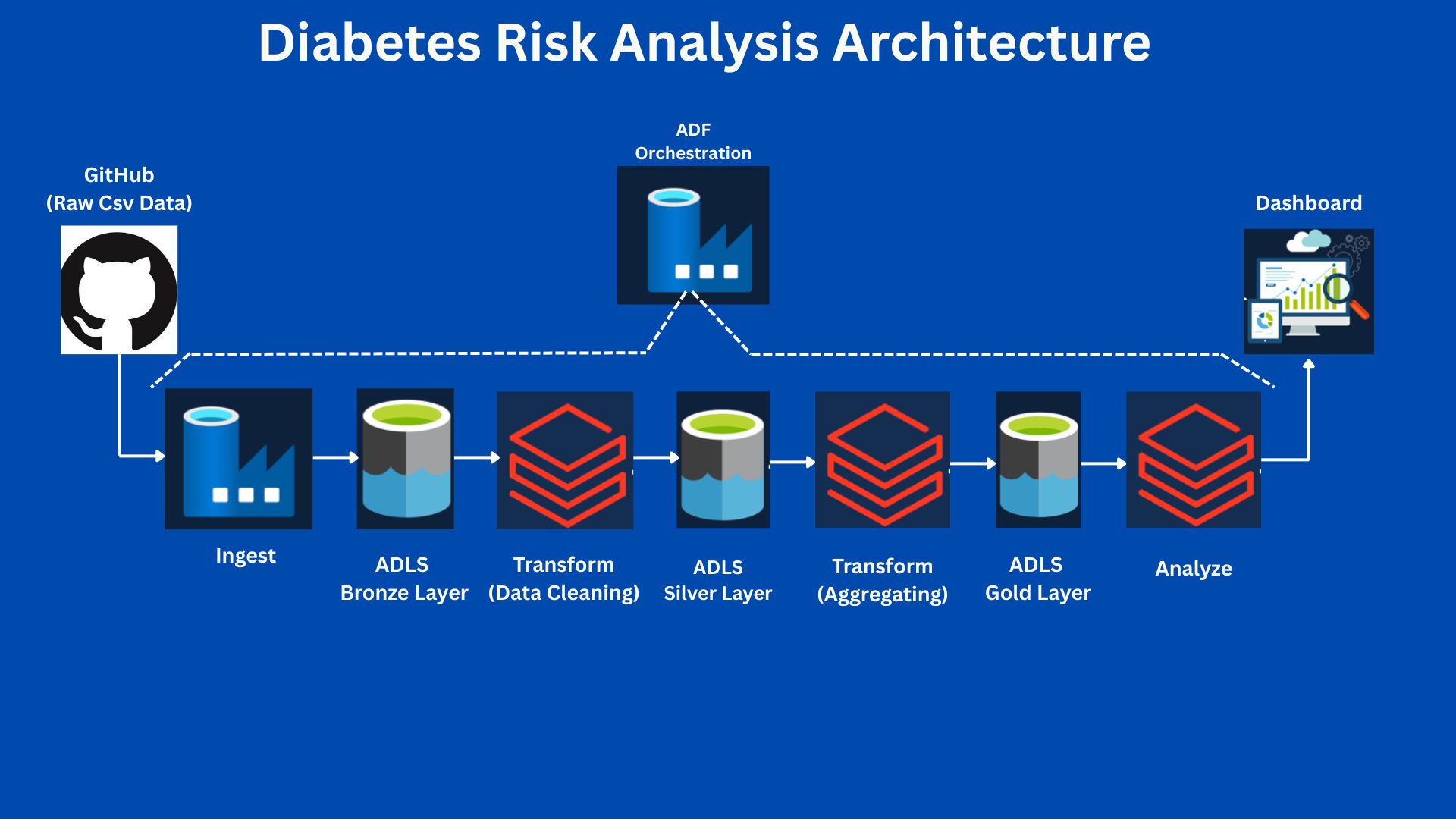
# **ETL Pipeline Documentation**

**Project Title:** Diabetes Risk Analysis ETL Pipeline  
**Course:** Data Engineering(CIS660-03)  
**Author:** Puneeth Kumar Amudala

**1.1 Overview**

This project simulates a real-world data engineering pipeline designed to transform raw health data into clean, structured, and analytics-ready datasets. The pipeline follows a modular ETL architecture and incorporates orchestration and cloud storage for end-to-end data flow. The pipeline is built using **Azure Data Factory** and **Azure Databricks** with data stored in **Azure Data Lake,** following the **Medallion Architecture**.



**1.2 Data Extraction – GitHub to Azure Data Lake (Bronze Layer)**

**1.2.1 Source**

The dataset was extracted from My GitHub repository containing health-related CSV files.

**1.2.2 Method**

* Created an HTTP linked service in Azure Data Factory to connect to GitHub
* Used a data pipeline in ADF to copy data into Azure Blob Storage (bronze container)
* Stored in path: wasbs://bronze@puneethdiabetesstorage.blob.core.windows.net/

**A screenshot of a computer

AI-generated content may be incorrect.**

**Deliverable:** Raw CSV file successfully ingested into Azure Data Lake (Bronze Layer).

**Reference:**  
*For detailed walkthrough, refer to:*  
👉 **“phase-1”**

**1.3 Data Transformation – Bronze to Silver (Phase 2)**

**1.3.1 Objective**

Transform raw CSV data into a cleaned and structured format stored in Parquet format in the Silver Layer.

**1.3.2 Steps**

1. **Mounting Bronze Layer in Databricks:**

dbutils.fs.mount(

source="wasbs://bronze@puneethdiabetesstorage.blob.core.windows.net/",

mount\_point="/mnt/diabetes/bronze",

extra\_configs={

"fs.azure.account.key.puneethdiabetesstorage.blob.core.windows.net": "my\_storage-key"

}

)

1. **Reading and Applying Schema:**

diabetes\_schema = StructType([...])

df\_raw = spark.read.csv("dbfs:/mnt/diabetes/bronze/\*.csv", header=True, schema=diabetes\_schema)

1. **Data Cleaning:**

* Removed duplicate values  
  df= df\_raw.dropDuplicates()
* Filtered out outliers  
  df\_filtered= df.filter(col('BMI')< 80)

1. **Write to Silver Layer:**

df\_filtered.write.format("parquet").mode("overwrite").save("/mnt/diabetes/silver/diabetes\_cleaned")

**Deliverable:** Cleaned Parquet data written to Azure Data Lake (Silver Layer).

**Reference:**  
*For detailed and cleaning logic, refer to:*  
👉 **“phase-2 and phase-3.docx”** and **“bronze\_to\_silver.html”**

**1.4 Data Enrichment and Aggregation – Silver to Gold (Phase 3)**

**1.4.1 Objective**

Enhance cleaned data with new features and perform aggregation by demographic segments. Final output is stored in Delta format in the Gold Layer.

**1.4.2 Transformations**

* **Age Grouping:**

df\_silver = df\_silver.withColumn("AgeGroup",

when(col("Age").isin("1.0", "2.0", "3.0"), "18-34")

.when(col("Age").isin("4.0", "5.0", "6.0"), "35-49")

.when(col("Age").isin("7.0", "8.0", "9.0"), "50-64")

.when(col("Age").isin("10.0", "11.0", "12.0", "13.0"), "65+")

.otherwise("Unknown")

)

* **Feature Engineering:**

df\_silver = df\_silver \

.withColumn("Obese", when(col("BMI") >= 30, 1.0).otherwise(0.0)) \

.withColumn("HighMentalDistress", when(col("MentHlth") > 15, 1.0).otherwise(0.0)) \

.withColumn("HighPhysicalDistress", when(col("PhysHlth") > 15, 1.0).otherwise(0.0)) \

.withColumnRenamed("HvyAlcoholConsump", "AlcoholRisk")  
  
  
df\_silver = df\_silver.withColumn("WeightedRiskScore",

(col("GenHlth") \* (0.277 / total\_weight)) +

(col("HighBP") \* (0.254 / total\_weight)) +

(col("BMI") \* (0.207 / total\_weight)) +

(col("DiffWalk") \* (0.205 / total\_weight)) +

(col("HighChol") \* (0.195 / total\_weight)) +

(col("Obese") \* (0.188 / total\_weight)) +

(col("HeartDiseaseorAttack") \* (0.168 / total\_weight)) +

(col("PhysHlth") \* (0.156 / total\_weight))

)

df\_silver = df\_silver.withColumn("RiskLevel",

when(col("WeightedRiskScore") > 5, "High")

.when((col("WeightedRiskScore") >= 3) & (col("WeightedRiskScore") <= 5), "Moderate")

.otherwise("Low")

)

* **Aggregation:**

df\_gold = df\_silver.groupBy("AgeGroup", "Sex", "RiskLevel").agg(

count("\*").alias("TotalPatients"),

round(avg("Diabetes\_binary"), 3).alias("DiabetesRate"),

round(avg("Obese"), 3).alias("ObesityRate"),

round(avg("PhysActivity"), 3).alias("LowActivityRate"),

round(avg("AlcoholRisk"), 3).alias("AlcoholRiskRate"),

round(avg("HighMentalDistress"), 3).alias("MentalDistressRate"),

round(avg("HighBP"), 3).alias("HighRiskRate")

)

1. **Write to Gold Layer:**

df\_gold.write.format("delta").mode("overwrite").partitionBy("AgeGroup").save("/mnt/diabetes/gold/")

**Deliverable:** Aggregated Delta table written to Gold Layer.

**Reference:**  
*For detailed and cleaning logic, refer to:*  
👉 **“phase-2 and phase-3”** and **“silver\_to\_gold.html”**

**1.5 Verification Queries**

* Example:

SELECT COUNT(\*) FROM gold\_vw;

A screenshot of a computer

AI-generated content may be incorrect.

* **Screenshot/Proof:** Included in separate verification document(Database/Storage Proof\_and\_Orchestration).

**1.6 Dashboard and Visualization (Phase 4)**

This phase used Databricks to generate simple visual summaries from the Gold-layer data.

**1.6.1 Objective**

To highlight key risk trends and behaviors using built-in charts.

**1.6.2 Highlights**

**Dashboard-1:**

**A screenshot of a medical report

AI-generated content may be incorrect.**

**Dashboard-2:**

A screenshot of a graph

AI-generated content may be incorrect.

* Pie chart: Patient distribution by age group
* KPI: Patient count by sex
* Scatter plot: Diabetes vs high-risk rate (by gender)
* Bar chart: Diabetes rate by age and sex
* Additional charts: Obesity, alcohol risk, and physical inactivity by demographic group
* Visualizations were created using built-in **Databricks notebooks** and display() functions.
* Interactive charts and summary tables were generated from Gold-layer Delta tables.

**1.6.4 Cloud Deployment (Optional)**

* No external deployment was done. All visualizations were performed within Databricks.

**Deliverable:** Dashboard visualizations were created and displayed within Databricks notebooks.

**1.7 Orchestration**

Azure Data Factory (ADF) was used to manage and automate the ETL flow. It handled:

* Downloading data from GitHub
* Running Databricks notebooks for each ETL phase
* Monitoring each step in the process

This setup made the workflow smooth, repeatable, and easy to track.

A screenshot of a computer

AI-generated content may be incorrect.

**Reference:**  
*For detailed and cleaning logic, refer to:*  
👉 **“phase-4” “gold\_to\_dashboard.html”**

**1.8 Best Practices Followed**

* Code modularity with structured notebooks for each phase
* Logging via notebook outputs and Azure logs
* Schema validation and cleaning logic enforced
* Used Delta format and Parquet for optimized storage and querying
* Developed reusable transformation steps and standardized visual storytelling

**1.9 Future Scalability**

* Can be extended with Kafka for real-time streaming
* Orchestrated using ADF; can be moved to Airflow or Kestra
* Supports large-scalex deployment using Databricks' distributed engine