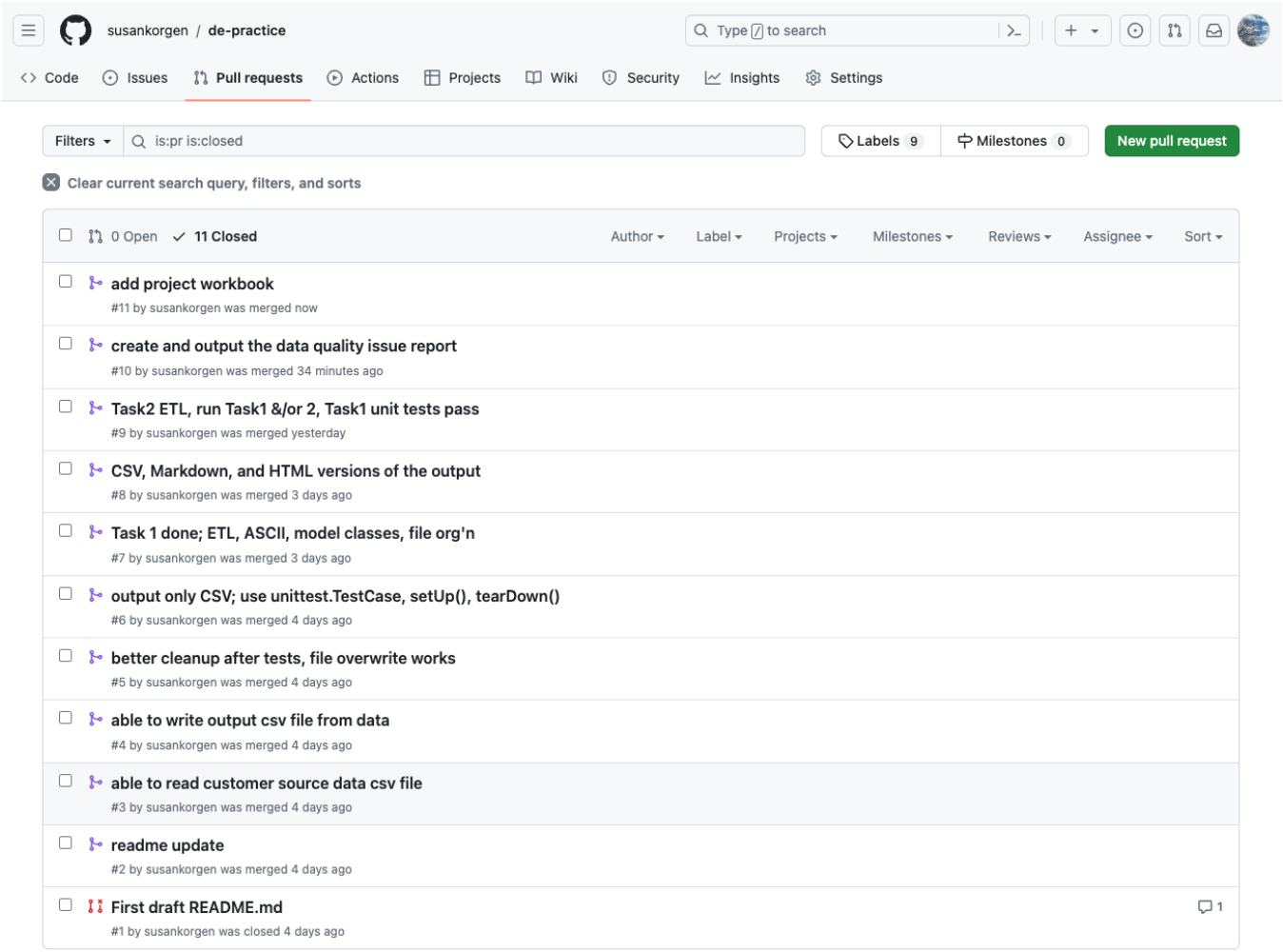
Assessment Workbook

Table of Contents

Summary



Context

I have 7+ years of expertise in data transformations and data modeling for healthcare. I am expert in FHIR (Fast Healthcare Interoperability Resources), C-CDA, X12, HL7, and others. Prior experience in the healthcare software space was more focused on products than on data.

I used Python, SQL, OCI, and Kotlin at my most recent job at Project Ronin for 2.5 years. There the **Data Platforms** team used Databricks and Spark. **Data Science** provided the models and algorithms

that consumed data. The data flow was Integrations > Data Platforms > Data Science, guided and informed by Informatics at pivot points in the pipeline. I worked on these two teams:

1. **Integrations** (1.5 years) wrangled external APIs at customer sites, managed initial data ingestion and validation, modeled standard FHIR, and transformed from FHIR to the Ronin data model for storage in the datalake. I participated in all team functions

2. **Informatics** (1 year), a larger team, decided the Ronin data model, provided mappings and tools for handling coded values in terminologies such as LOINC and SNOMED, and flattened tables for final pipeline steps before Data Science. I extended the terminology mapping system with a new pipeline validation workflow, in which new mappings corrected failures reported by the data validation server built by the Integrations team. I also designed and implemented an overhaul of our terminology mappings database to support new types of FHIR data structures with complete flexibility, so that we no longer needed a database migration per each model change to support corporate product needs.

Day 1

Project Requirements, Success Factors, Constraints

Extracted these priorities from writeup:

1. Immediate: **Get customer to sign quickly** (money starts coming in sooner)
    a. How is success measured:
        i. Demo our predictive models work and produce meaningful results.
2. After signing: **Optimize throughput per avocado** (more money comes in faster)
    a. How is success measured:
        i. processing time per avocado.

Constraints:

1. Transform input data into the format Data Science needs.
2. Results display must be understood by a stakeholder who reads SQL but not spark.
3. Create a repo and demonstrate peer review.

Start Workbook

This doc is a running log for my use. It is *optional* to read! There will be a presentation.

Sprint Plan: 4 Days, 1 Epic, 14 Stories

T1: Study all text in assessment to extract requirements and success factors
T2: Start a Workbook to gather ideas as I go (analogous to a Confluence space) e
T3: Task 1 input data model, output data model
T4: Design and implement Task 1 transformations
T5: Design Task 2 transformations
T6: Incorporate iteration in demo
T7: Meet with Jun to align on the exercise so far
T8: Data exploration, identify real and potential data quality issues, document a plan

T9: Implement Task 2 transformations
T10: Design basic pipeline (diagram) – S3 buckets, customer loads, trigger read, do ETL, write result
T11: Design final pipeline (diagram) – analytics/customers read result, add write/read of DQI log
T12: Implement final pipeline
T13: Add clone and demo instructions to README

Scrum

T1: study all text in assessment to extract requirements and success factors
T2: start a Workbook to gather ideas as I go (analogous to a Confluence space)
B1: Databricks: I have not created anything in Databricks, but I used existing notebooks at Ronin
B2: Cloud: read/wrote from/to OCI buckets using Python(writes) and Kotlin(reads) at Ronin
B3: Spark: Data Platforms used it at Ronin. I used Python in Informatics, Kotlin in Integrations

Databricks Knowledge

Plans: address lack of Databricks experience with:

1. short-term quick training measures to move the assignment forward.
2. **simply assume we will use Databricks**, since it was used at my previous employer
3. assignment mentions discussing Databricks vs. other options; that's appropriate for this test. If I can't do that satisfactorily with quick research, I can offer an answer on a similar question from my direct experience, comparing Python and Kotlin as languages for a back-end team, pros and cons, impact on productivity vs. stability.

Cloud Knowledge

Since I don't know AWS, for this work **I will simulate S3 buckets using the local file system**.

Seems fine, because after working with OCI (Oracle Cloud Infrastructure) for 2 years at Ronin:

1. for reads or writes, and
2. from my code written in either Python (for writes) or Kotlin (for reads), and
3. sending either JSON or CSV (depending on needs of the consumer for data format),
4. sometimes I was the consumer and sometimes I was the provider; I saw both sides, and
5. except for writing and then calling the service APIs for read/write, which I contributed to:
6. **daily use of OCI from my Python or Kotlin software simply felt like file system calls**.

Spark Knowledge

I have not needed to use Spark professionally, so where I might have used it, I will use Python.

Task 1

Quick and dirty demo started in the repo: **main.py** and **test_main.py** and supporting files.

Repo and Peer Review

Created GitHub repo: https://github.com/susankorgen/de-practice

I want to avoid clogging my process during a time-limited test, so other than disallow force push,

I am not configuring many restrictions. For teamwork in repos, I prefer more checks, like these:

1. Require a pull request before merging.
2. At least 2 approvals required (other than the author) before allowing merge.
3. GitHub actions exist checking code coverage, and other standards before allowing merge.
4. GitHub actions exist for deploy and release to "dev-stage-prod" staged environments.

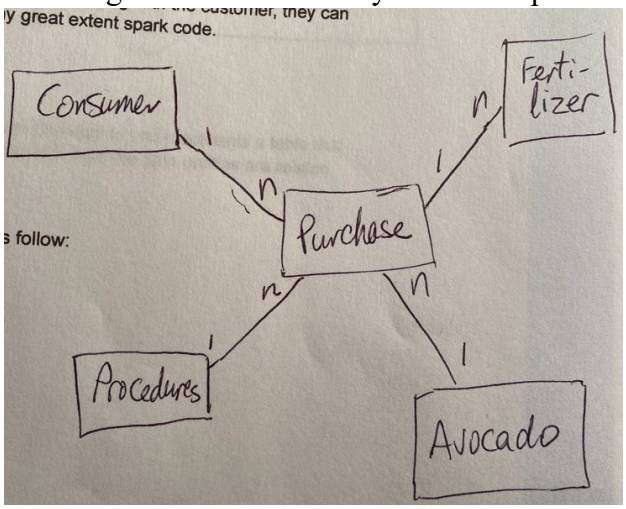
Best Practices

I provide a high level of code coverage in Task 1 Day 1 work, to demonstrate that I value and understand **unit tests** and add them along with each code update. Beyond that, I won't plan to attempt a professional level of unit testing but will focus on finishing the assignment.

Code comments: Python PEP standards advise docstrings only for non-obvious functions. I am famous for my docstrings for complex interrelated functions within custom framework code. People say they can follow and use these functions rapidly. Here I'm using restraint (deliberate!)

Customer Data Model (Diagram)

This diagram shows the entity relationships in the customer-provided model documentation.



Trust but Verify, or: Truth is in the Data

The **model** folder in repo has skeleton classes that I implemented during Task 1 to conform to the above diagram, as I expected it to be accurate. Later, **just like in real life**, while exploring the 4 CSV files for Task 2, I saw that the customer-provided documentation and diagram were wrong on important points.

Be nice about the customer docs you get, but **always get samples of data before moving forward**.

Day 2

Scrum

Y: PyCharm environment and GitHub setup, read/write CSV from Python with required params
T3: Task 1 input data model, output data model

T4: Design and implement Task 1 transformations
T5: Design Task 2 transformations
B4: (see below) how to best address Databricks for this assignment

Databricks Knowledge

I learn and apply the new tech required at work (see my LinkedIn Recommendations).

My teams at Ronin and InterSystems transformed data by making SQL queries, or via cloud buckets or file systems, or via vendor or internal APIs, whatever was needed, then transformed this data (iterated over resultsets or interpreted payloads) as objects (dynamic or structured) using purpose-built code framworks we wrote in Python, Kotlin, JavaScript, or ObjectScript.

I will use the above approach in my repo. This code will be easy to read and will achieve goals. To address the closing deliverables in the assessment, I will research and add in my presentation, remarks on how I would reimplement the same work using Databricks.

Task 1

Quick and dirty demo continued in the repo: **main.py** and **test_main.py** and supporting files.

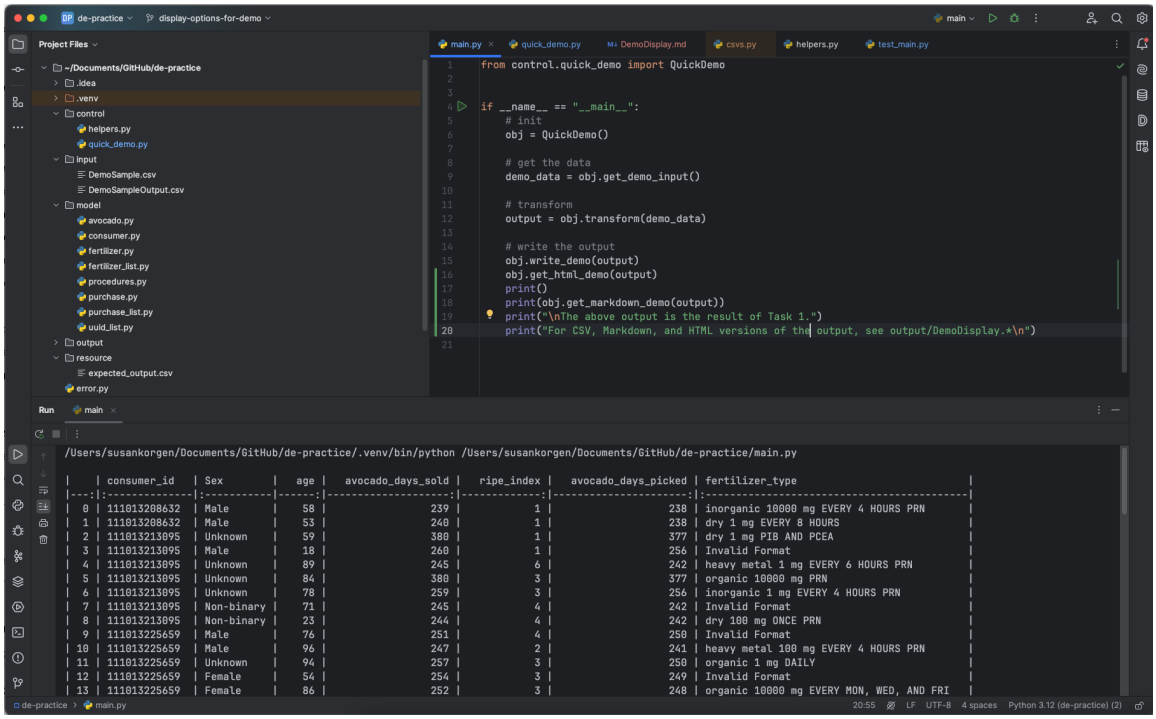
Task 1 ETL Plan

Transformation of 1 CSV input file to 1 CSV output file:

- consumer_id:
  - **consumerid** (non-unique string)
  - Best practice from healtcare IT is to use string for identifiers.
  - Even if ids look like float or int, in the wilds of HIT, sources mix character types.
  - > Consumer.uuid (UUID) and Consumer.consumerid (original value, as str)
  - > output.consumer_id (original value, as str)
- Sex:
  - note: HL7v2 four-choice gender is M|F|N|U, Male Female Non-binary Unknown
  - **Randomly** choose from a string list
  - > Consumer.sex (str)
  - > output.sex (str)
- age:
  - **Randomly** generate (int, range 1-105)
  - > Consumer. age (int)
  - > output.age (int)
- avocado_days_sold:
  - Pretend **avocado_bunch_id** in “quick input” is days from picked to sold (it isn’t)
  - avocado_days_picked (calculated) + avocado_bunch_id
  - > output.avocado_days_sold (int, range >= 0)
- ripe_index:
  - avocado_price_index (int)
  - Pretend **avocado_price_index** in “quick input” is a ripe index (it isn’t)
  - > output.ripe_index (original int)

- avocado_days_picked**:**
    - All dates are in 1998, so for demo, adjust to last year (2023)
    - (adjust datetime format) today minus **graphed_date** + 15Y, days (int)
    - > output.avocado_days_picked (int)
- fertilizer_type:
    - Mock fertilizer data for an interesting demo display, that is:
    - **Randomly** generate a combination of (fertilizer.type, .mg, .frequency) from input
    - > output.fertilizer_type (str)

Command Line Demo with tabulate Output



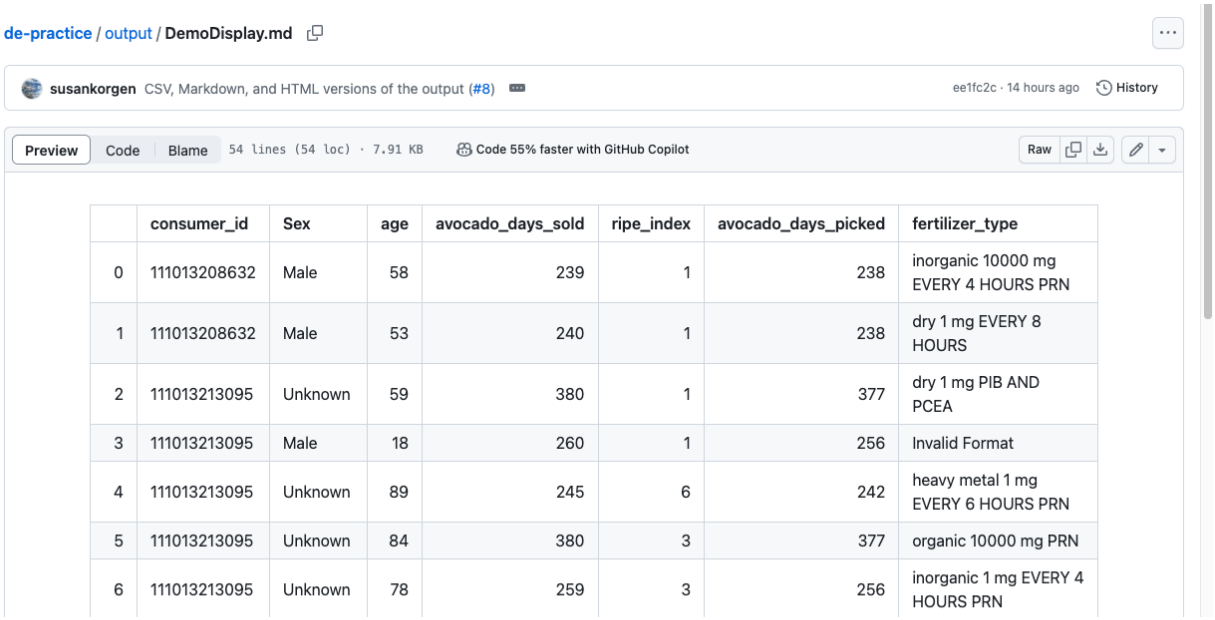
```
/Users/susankorgen/Documents/GitHub/de-practice/.venv/bin/python /Users/susankorgen/Documents/GitHub/de-practice/main.py

|    | consumer_id   | Sex        |   age |   avocado_days_sold |   ripe_index |   avocado_days_picked | fertilizer_type                          |
|---:|:--------------|:-----------|------:|--------------------:|-------------:|----------------------:|:-----------------------------------------|
|  0 | 111013208632  | Male       |    58 |                 239 |            1 |                   238 | inorganic 10000 mg EVERY 4 HOURS PRN     |
|  1 | 111013208632  | Male       |    53 |                 240 |            1 |                   238 | dry 1 mg EVERY 8 HOURS                   |
|  2 | 111013213095  | Unknown    |    59 |                 380 |            1 |                   377 | dry 1 mg PIB AND PCEA                    |
|  3 | 111013213095  | Male       |    18 |                 260 |            1 |                   256 | Invalid Format                           |
|  4 | 111013213095  | Unknown    |    89 |                 245 |            6 |                   242 | heavy metal 1 mg EVERY 6 HOURS PRN       |
|  5 | 111013213095  | Unknown    |    84 |                 380 |            3 |                   377 | organic 10000 mg PRN                     |
|  6 | 111013213095  | Unknown    |    78 |                 259 |            3 |                   256 | inorganic 1 mg EVERY 4 HOURS PRN         |
|  7 | 111013213095  | Non-binary |    71 |                 245 |            4 |                   242 | Invalid Format                           |
|  8 | 111013213095  | Non-binary |    23 |                 244 |            4 |                   242 | dry 100 mg ONCE PRN                      |
|  9 | 111013225659  | Male       |    76 |                 251 |            4 |                   250 | Invalid Format                           |
| 10 | 111013225659  | Male       |    96 |                 247 |            2 |                   241 | heavy metal 100 mg EVERY 4 HOURS PRN     |
| 11 | 111013225659  | Unknown    |    94 |                 257 |            3 |                   250 | organic 1 mg DAILY                       |
| 12 | 111013225659  | Female     |    54 |                 254 |            3 |                   249 | Invalid Format                           |
| 13 | 111013225659  | Female     |    86 |                 252 |            3 |                   248 | organic 10000 mg EVERY MON, WED, AND FRI |
```

Display Options for Lead Business Stakeholder (and Customer)

Output Markdown and HTML as display options for the lead business stakeholder and customer.

Markdown Output



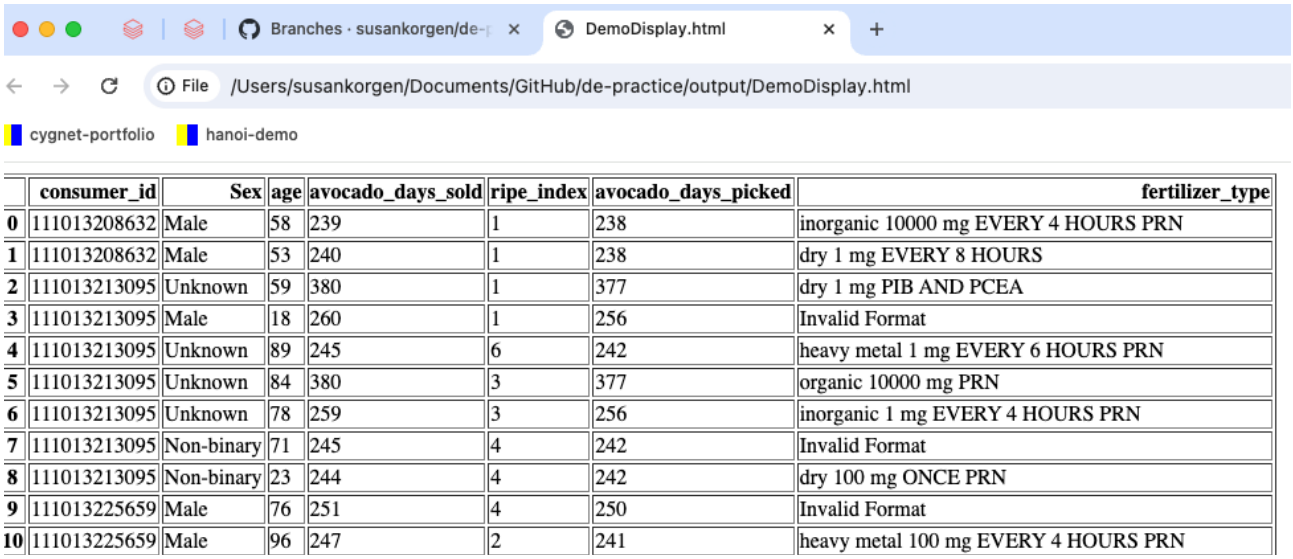
de-practice / output / DemoDisplay.md

susankorgen CSV, Markdown, and HTML versions of the output (#8) ee1fc2c · 14 hours ago History

Preview Code Blame 54 lines (54 loc) · 7.91 KB Code 55% faster with GitHub Copilot Raw

	consumer_id	Sex	age	avocado_days_sold	ripe_index	avocado_days_picked	fertilizer_type
0	111013208632	Male	58	239	1	238	inorganic 10000 mg EVERY 4 HOURS PRN
1	111013208632	Male	53	240	1	238	dry 1 mg EVERY 8 HOURS
2	111013213095	Unknown	59	380	1	377	dry 1 mg PIB AND PCEA
3	111013213095	Male	18	260	1	256	Invalid Format
4	111013213095	Unknown	89	245	6	242	heavy metal 1 mg EVERY 6 HOURS PRN
5	111013213095	Unknown	84	380	3	377	organic 10000 mg PRN
6	111013213095	Unknown	78	259	3	256	inorganic 1 mg EVERY 4 HOURS PRN

HTML Output



Branches · susankorgen/de- DemoDisplay.html

File /Users/susankorgen/Documents/GitHub/de-practice/output/DemoDisplay.html

cygnet-portfolio hanoi-demo

	consumer_id	Sex	age	avocado_days_sold	ripe_index	avocado_days_picked	fertilizer_type
0	111013208632	Male	58	239	1	238	inorganic 10000 mg EVERY 4 HOURS PRN
1	111013208632	Male	53	240	1	238	dry 1 mg EVERY 8 HOURS
2	111013213095	Unknown	59	380	1	377	dry 1 mg PIB AND PCEA
3	111013213095	Male	18	260	1	256	Invalid Format
4	111013213095	Unknown	89	245	6	242	heavy metal 1 mg EVERY 6 HOURS PRN
5	111013213095	Unknown	84	380	3	377	organic 10000 mg PRN
6	111013213095	Unknown	78	259	3	256	inorganic 1 mg EVERY 4 HOURS PRN
7	111013213095	Non-binary	71	245	4	242	Invalid Format
8	111013213095	Non-binary	23	244	4	242	dry 100 mg ONCE PRN
9	111013225659	Male	76	251	4	250	Invalid Format
10	111013225659	Male	96	247	2	241	heavy metal 100 mg EVERY 4 HOURS PRN

Task 2

More realistic demo with more data, started in the repo: **main.py** and **test_main.py** and supporting files.

Task 2 ETL Plan

Transformation of 4 CSV input files to 1 CSV output file:

- consumer_id:
  - **consumer.consumerid** (non-unique float)
  - As an identifier, it doesn't matter this value looks like a floating point number.
  - The meaning is the same (the same values will match) if we use string for it.
  - If we read it as a float we could actually introduce errors depending on platform.
  - > Consumer.uuid (UUID), Consumer.consumerid (original value, as string)
  - > output.consumer_id (original value, as string)
- Sex:
  - note: HL7v2 four-choice gender is M|F|N|U, Male Female Non-binary Unknown
  - We see that the only values are Male, Female now, but an enum could be limiting.
  - **consumer.Sex** (string)
  - > Consumer.sex (original string)
  - > output.Sex (original string)
- age:
  - **consumer.age** (int, range >= 0)
  - > Consumer. age (original int)
  - > output.age (original int)
- avocado_days_sold:
  - (adjust datetime format) **avocado.sold_date** minus **avocado.born_date** days (int)
  - > output.avocado_days_sold (int, range >= 0)
- ripe_index:
  - **avocado."ripe index when picked"** (int, range 0-10)
  - > output.avocado_ripe_index (original int)
- avocado_days_picked:
  - (adjust datetime format) **avocado.sold_date** minus **avocado.picked_date** days
  - > output.avocado_days_picked (int, range >= 0)
- fertilizer_type:
  - (list of str) all **fertilizer.type** (enum) for the same **fertilizer.purchaseid**
  - > use distinct values found and join with a comma
  - > we could get fancier if it was for display rather than analysis, e.g. "type (occurences)"
  - > a CSV list in the output column

Day 3

Scrum

Y: Task 1 ETL code, ASCII format issue, output CSV per spec, and HTML/md for nice demo
T7: Meet with Jun to align on the exercise so far
T8: Data exploration, identify real and potential data quality issues, document a plan
T5: (continued) Design Task 2 transformations

Task 2

1. Plan ETL: we did that on Day 2.
2. Read in the 4 new csv files.
3. Implement ETL.
4. Show output files like the previous example
5. Revise the "random" demo helpers to set a flag to use static data for unit tests.
6. Make a data quality report – don't weight the issues by type yet – see list below

Real-World Data

As is typical for real-world data:

1. This sample input has many fields and values our scenario does not need.
2. It is well worth doing the ETL upon ingestion, to:
    a. Limit our stored data to the values our cases need.
    b. Reduce storage costs and make data extraction faster and easier.

Data Quality Issues

We want to work around, report to a log, and/or analyze for a score, any input data quality issues.

Certain data quality issues can totally block data analysis for a customer, especially problems with primary keys (PK) and foreign keys (FK) in the customer data. An example of PK issue in the source data is that both of the following are true. As a result, we can't analyze any consumer details, yet that is the most important thing to analyze for our customer!

1. consumerid is a PK in **consumer**, but consumerid values are not unique in **consumer**
2. Because each consumerid value matches multiple rows in **consumer**, we cannot pivot to other tables to support analysis based on on any of the consumer fields such as sex or age. We can't do anything for our customers, so this is unacceptable, a showstopper.
3. Side note, FKs: only one consumerid value appears in every row of **avocado**, **fertilizer**, and **purchase**. That would be odd in 24 hours and should be flagged as a potential issue.

Data Exploration can reveal DATA DESIGN Quality Issues

Data design quality issues we need to discover while exploring customer data to work out the join pivots for queries on that data. In real life, we need to discuss **data design** quality issues with the customer during data exploration, before implementing ETL code, while onboarding them as a newcomer who contracted to use our service. For the exercise, assume all is confirmed.

Duplicate PK names in model

Avocado and purchase both have PKs named purchaseid in the customer model doc. Bad sign…

Positive int vs. int

Many columns are counts of objects and can be constrained to positive int. **FHIR does this often.**

Column being used as a PK not identified as a PK, and so-called PK is really a FK

We want to join other tables with **avocado** but the values in its identified PK purchaseid are not unique identifiers, and are actually being used as a FK into **purchase** in the sample data.

avocado column avocado_bunch_id values are all unique. It works like a PK in **avocado** and works like a FK in other tables. Inspection of CSV files shows all purchase.avocado_bunch_id values are from avocado.avocado_bunch_id.

Will correct our copy of the customer model to match the model implied by the sample data:

1. avocado.avocado_bunch_id is a PK
2. avocado.purchaseid is a FK
3. purchase.avocado_bunch_id is a FK

2 FKs where 1 FK could work

The customer doc says **fertilizer** has FKs purchaseid and consumerid. Yes, **fertilizer** has these columns, but this model feels large and clunky. It was not wrong to provide these two FKs, because the question of "organic" vs. other fertilizers might need to be analyzed relative to consumers and purchases, but:

For the data model, it seems cleaner and simpler to:

- provide 1 FK from fertilizer to avocado, which we know we always want, then analyze purchase and consumer details as needed using those 2 FKs from avocado.

Rather than:

- provide the 2 weak, indirect FKs from fertilizer to purchase and consumer while requiring every query to pivot and extra hop from fertilizer to the more relevant avocado.

The relationship we want is the direct one between **fertilizer** and **avocado**.
However, will not correct our copy of the customer model at this time. Will pivot the extra hop:

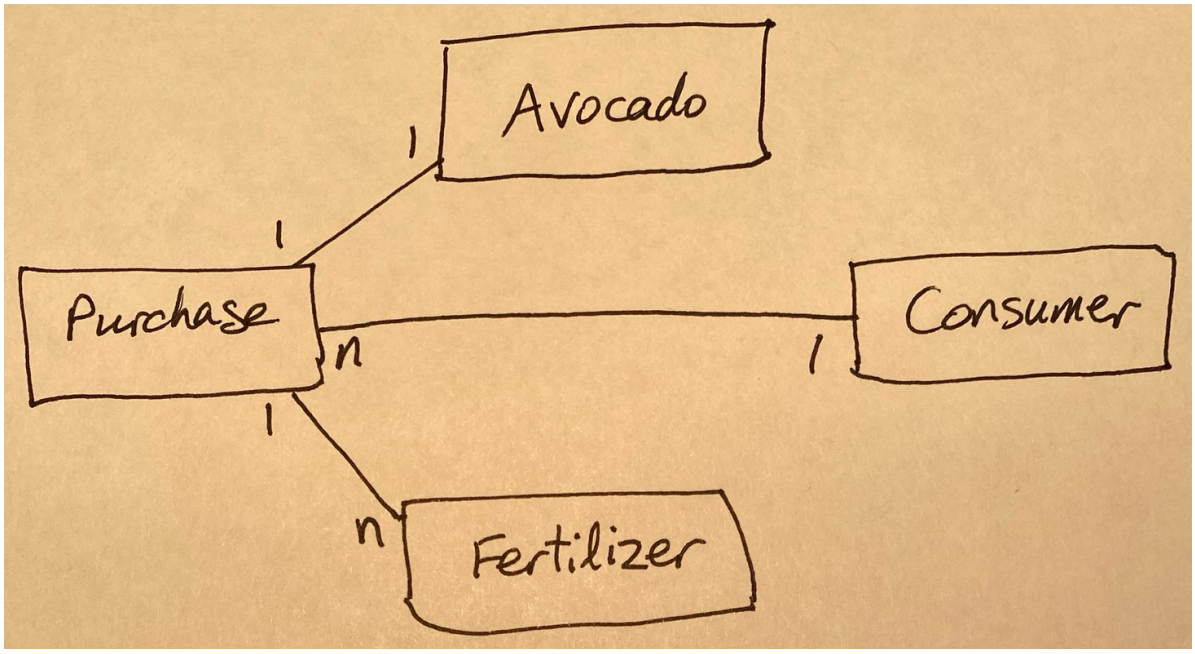
1. fertilizer.purchaseid is a FK from **fertilizer** to **purchase**
2. purchase.avocado_bunch_id is a FK from **purchase** to **avocado**

Bad PKs in 1 table

Consumer.consumerid sample data values are quite bad, a BLOCKER issue (see issue #1 below).
In real life, while first onboarding this customer after they signed on with us, then either:

1. they must correct their data to provide all distinct values in consumer.consumerid, or
2. they must to sacrifice these benefits:
    a. receiving any consumer values in their output, and
    b. using any consumer data in their analysis.

Customer Data Model Diagram, Corrected



Customer Data Model Doc, Corrected

Table Fertilizer:

fertilizerid	NOT NULL PK	
consumerid	NOT NULL FK	→ consumer
purchaseid	NOT NULL FK	→ purchase

Table Purchase:

purchaseid	NOT NULL PK	
consumerid	NOT NULL FK	→ consumer
avocado_bunch_id	NOT NULL FK	→ avocado

Table Avocado:

avocado_bunch_id	NOT NULL PK	
purchaseid	NOT NULL FK	→ purchase
consumerid	NOT NULL FK	→ consumer

Table Consumer:

consumerid (all values bad)	NOT NULL PK	

Pivot Plan for ETL

Iterating over the avocado rows in each data load, we get data from avocado, consumer, and fertilizer:

- avocado joins purchase on avocado_bunch_id
- purchase joins fertilizer on purchaseid (gets >1 fertilizer rows per purchaseid, select distinct)
- purchase joins consumer on consumerid (but data is so bad, for demo select random comsumer)

ETL Daily Runs can reveal DATA VALUE Quality Issues

Data value quality issues are discoverable during daily processing. From most to least severe:

1. BLOCKER – for the exercise, will report and work around; in real life these must be fixed:
    a. PK is not unique in table (consumerid) – (for demo: mock values)
    b. Missing value for a field that is NOT NULLABLE (PKs and FKs) – (for demo: skip row)
2. MISSING – (BLOCKER is also reported for these) Missing PK or FK for our scenario:
    a. **consumer.consumerid**
    b. **fertilizer.purchaseid**
    c. **fertilizer.consumerid**
3. MISSING – Missing value for a data field needed for our scenario, demo substitutes:
    a. **consumer.Sex** (for demo: random value)
    b. **consumer.age** (for demo: random value)
    c. **avocado.sold_date** (for demo: make today)
    d. **avocado.born_date** (for demo: make today)
    e. **avocado."ripe index when picked"** (for demo: make 0)
    f. **avocado.picked_date** (for demo: make today)
4. DATATYPE – will flag and discuss with customer; for demo, will try to adjust data type
    a. Inconsistent date format values (for demo: strip time from date strings)
    b. Inappropriate data type for identifier (float for any “id”, should be int or str)
    c. Date value that is non-empty but not parseable as a date. (for demo: empty)
5. RANGE – will flag; can help customer decide on path forward; for demo, will adjust
    a. Super old dates (for demo: set to 365)
    b. Negative ages (for demo: make 0)
    c. Negative numbers of days (for demo: make 0)
    d. Negative ripe index (for demo: make 0)
    e. Ripe index > 10 or < 0 (for demo: adjust to 10 or 0 respectively)
    f. Dates in the future as of today (for demo: make today)
6. UNEXPECTED – flag it; can help customer decide on path forward; for demo, adjust
    a. same FK value in 95-100% of rows (consumerid) (for demo: random value)
    b. non-ASCII char in input but ASCII output is required (for demo: “Invalid”)
    c. NaN in a float (for demo: 0.0)
    d. NaN in an int (for demo: 0)
7. NONSTANDARD – will flag; could discuss with customer, but often they can’t change
    a. Spaces in a field name: avocado."ripe index when picked"
    b. Capitalized column names: Sex, Race Age in consumer.csv
    c. Capitalized column name in output spec: Sex

Day 4

Scrum

Y: Data exploration, confirm requirements with Jun, plan data quality, design Task 2 transformations
T9: Implement Task 2 transformations
T10: Design basic pipeline (diagram) –S3 buckets, customer loads, trigger read, do ETL, write result
T11: Design final pipeline (diagram) – analytics/customers read result, add write/read of DQI log
T12: Implement final pipeline
T13: Add clone and demo instructions to README
T6: (Ran out of time) Incorporate iteration in demo
B5: repo is using DataFrame (for easy CSV read/write) – have not used queries with DataFrame before

DataFrame, or: Extending a Successful Prototype to Make a Product

In Task 1, I focused on quickly implementing CSV read/write, so I used pandas DataFrame. Now in Task 2, I want queries to select data. For queries, I have used only SQL. But I haven't got an SQL server set up for this project and it doesn't make sense to do it now. **Just as in real life**, a choice I made for my "successful prototype" (Task 1) is now a blocker for my "product" phase (Task 2).

To save time, I wrote SQL (see docstring in ETLDemo.transform()) and asked ChatGPT to translate.

AI Helpers for Work

You won't see any ChatGPT executable code in my repo. I do use ChatGPT as a sounding board while working. ChatGPT answers usualy contain hints that inspire me to choose a useful next step for myself. I don't give ChatGPT proprietary information or identifying details for any problem in my chats with it.

When my employer has a policy about AI Helpers, I follow it.

Protected Health Information (PHI)

I handle PHI carefully [er guidelines from HIPAA and other standards. I never use PHI in an AI helper.

For demos, I either

1. de-identify data,
2. use fictional/generated data whose origin is me or a verified source (Synthea, vendor sandboxes)
3. get permission from data stewards to show PHI on a need-to-know basis with members of the organization that have permission to view the PHI in that context (etc, per operative standards)

Task 2

More realistic demo with more data, continued in the repo: **main.py**, **test_main.py** and supporting files.

Task 2 ETL Implementation

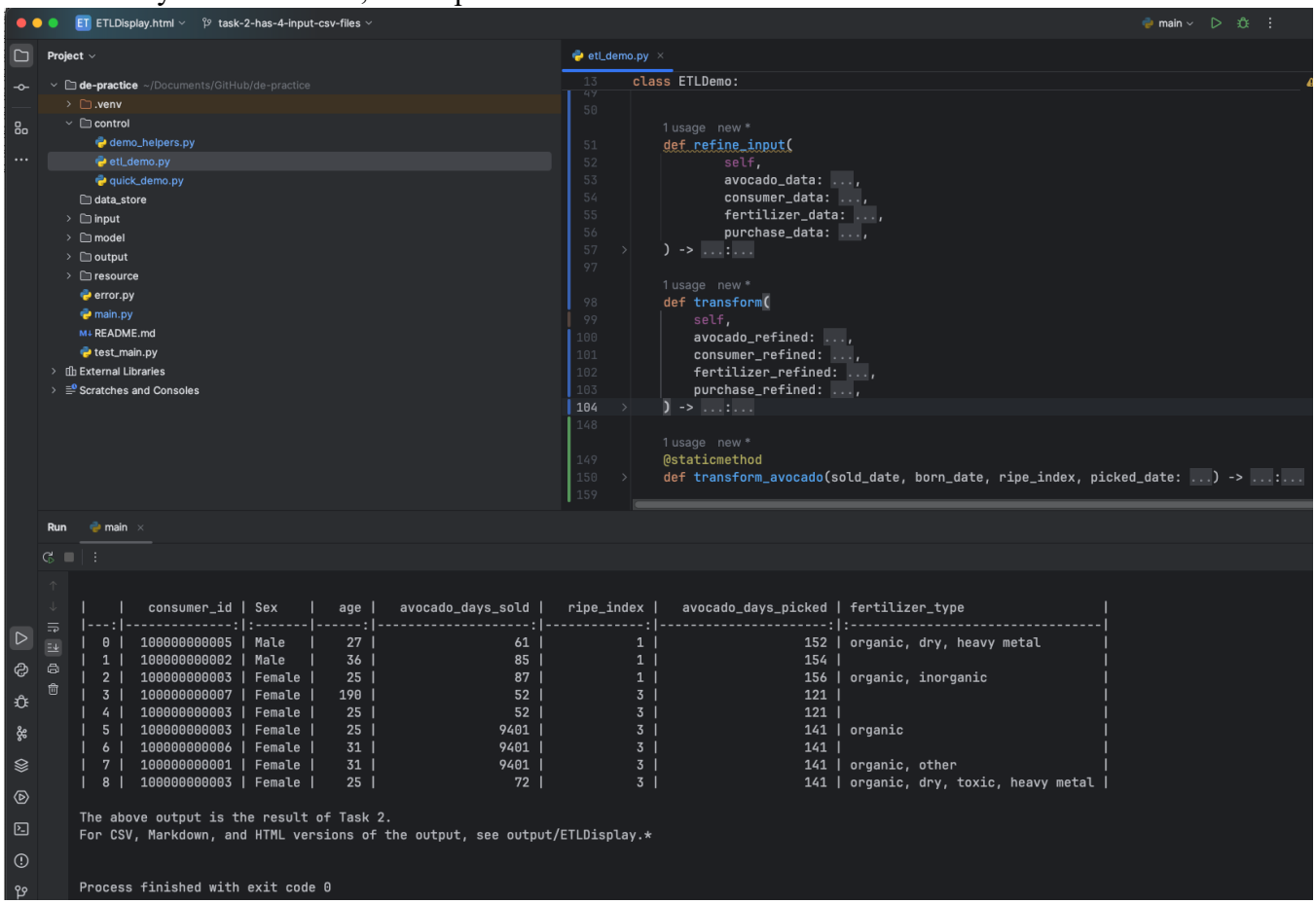
In the repo.

Command Line Demo with tabulate output

The Task 2 ETL implementation:

1. outputs the CSV file to the **output** folder
2. on the screen, displays Task1 output followed by Task 2 output, formatted using **tabulate**
3. outputs Markdown and HTML for easy viewing the lead business stakeholder and customer.

As seen in PyCharm console, lower panel.



Display Options for Lead Business Stakeholder (and Customer)

Markdown and HTML (next page).

Markdown Output

As seen in GitHub repo.

	consumer_id	Sex	age	avocado_days_sold	ripe_index	avocado_days_picked	fertilizer_type
0	100000000003	Female	25	61	1	152	organic, dry, heavy metal
1	100000000001	Female	31	85	1	154	
2	100000000002	Male	36	87	1	156	organic, inorganic
3	100000000007	Female	190	52	3	121	
4	100000000003	Female	25	52	3	121	
5	100000000005	Male	27	9401	3	141	organic
6	100000000002	Male	36	9401	3	141	
7	100000000005	Male	27	9401	3	141	organic, other
8	100000000001	Female	31	72	3	141	organic, dry, toxic, heavy metal

Leading to the next topic, this output shows:

1. some data values (**customer_id**) have been adjusted or faked to overcome BLOCKING issues.
2. others would benefit from adjustment as they are clearly wrong (some **age** values)
3. In real life, as discussed in Day 3 in this Workbook:
    a. we would detect log and label data quality issues of specific types and levels
    b. we would work with the customer to resolve all issues and agree on adjustments (if any)

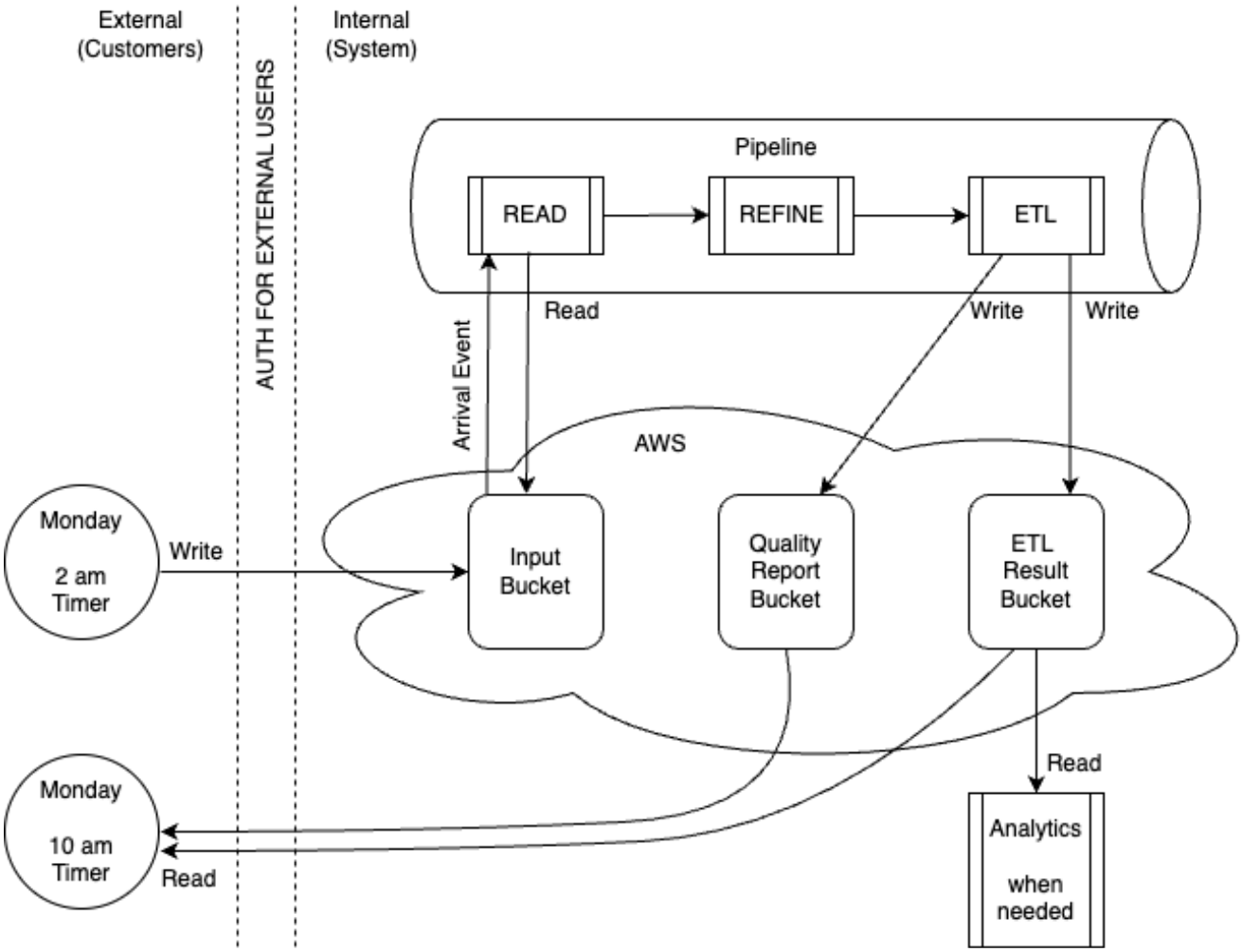
HTML Output

	consumer_id	Sex	age	avocado_days_sold	ripe_index	avocado_days_picked	fertilizer_type
0	100000000003	Female	25	61	1	152	organic, dry, heavy metal
1	100000000002	Male	36	85	1	154	
2	100000000007	Female	190	87	1	156	organic, inorganic
3	100000000007	Female	190	52	3	121	
4	100000000002	Male	36	52	3	121	
5	100000000005	Male	27	9401	3	141	organic
6	100000000006	Female	31	9401	3	141	
7	100000000007	Female	190	9401	3	141	organic, other
8	100000000002	Male	36	72	3	141	organic, dry, toxic, heavy metal

Data Pipeline Architecture (Diagram)

Pipeline components to implement and demo:

- S3 buckets (demo simulate using file folders on local file system)
- reads/writes from external customers go through an extra authentication barrier
- customer loads input on a timer trigger, Mondays at 2am
- input arrival event triggers the pipeline read segment to start (we will trigger manually)
- inside pipeline, the 3 segments are READ > REFINE > ETL (gathering DQI along the way)
- at end, ETL writes the result and DQI to their respective buckets
- customer reads result on a timer trigger, Mondays at 10am
- customer may ask to read DQI as needed
- analytics services may ask to read result as needed



Data Quality Implementation

In the repo, see the new **quality** folder and its Python files for the code.

Data Quality Issue Report

HTML Example

	name	severity	description	demo_workaround	pipeline_segment	table_name	field_name
558	BLOCKER-1a	50.00	PK is not unique in table	mock values	transform	consumer	consumerid
562	BLOCKER-1a	50.00	PK is not unique in table	mock values	transform	consumer	consumerid
566	BLOCKER-1a	50.00	PK is not unique in table	mock values	transform	consumer	consumerid
569	BLOCKER-1a	50.00	PK is not unique in table	mock values	transform	consumer	consumerid
573	BLOCKER-1a	50.00	PK is not unique in table	mock values	transform	consumer	consumerid
577	BLOCKER-1a	50.00	PK is not unique in table	mock values	transform	consumer	consumerid
581	BLOCKER-1a	50.00	PK is not unique in table	mock values	transform	consumer	consumerid
585	BLOCKER-1a	50.00	PK is not unique in table	mock values	transform	consumer	consumerid
588	BLOCKER-1a	50.00	PK is not unique in table	mock values	transform	consumer	consumerid
559	MISSING-3a	25.00	Missing value for a field required for demo	choose random row from real data	transform	consumer	consumerid
563	MISSING-3a	25.00	Missing value for a field required for demo	choose random row from real data	transform	consumer	consumerid
567	MISSING-3a	25.00	Missing value for a field required for demo	choose random row from real data	transform	consumer	consumerid
570	MISSING-3a	25.00	Missing value for a field required for demo	choose random row from real data	transform	consumer	consumerid
574	MISSING-3a	25.00	Missing value for a field required for demo	choose random row from real data	transform	consumer	consumerid
578	MISSING-3a	25.00	Missing value for a field required for demo	choose random row from real data	transform	consumer	consumerid
582	MISSING-3a	25.00	Missing value for a field required for demo	choose random row from real data	transform	consumer	consumerid
586	MISSING-3a	25.00	Missing value for a field required for demo	choose random row from real data	transform	consumer	consumerid
589	MISSING-3a	25.00	Missing value for a field required for demo	choose random row from real data	transform	consumer	consumerid
572	RANGE-5a	8.00	Value too low, out of range	set to minimum in range	transform	consumer	Age
580	RANGE-5a	8.00	Value too low, out of range	set to minimum in range	transform	consumer	Age
561	RANGE-5b	7.50	Value too high, out of range	set to maximum in range	transform	consumer	Age
565	RANGE-5b	7.50	Value too high, out of range	set to maximum in range	transform	consumer	Age
576	RANGE-5b	7.50	Value too high, out of range	set to maximum in range	transform	consumer	Age
584	RANGE-5b	7.50	Value too high, out of range	set to maximum in range	transform	consumer	Age
557	UNEXPECTED-6b	4.75	Non-ASCII char in str but ASCII output is required	use 'other'	refine_input	fertilizer	type
1	UNEXPECTED-6i	0.39	float where an int should be	cast as int	refine_input	consumer	consumerid
2	UNEXPECTED-6i	0.39	float where an int should be	cast as int	refine_input	consumer	consumerid
3	UNEXPECTED-6i	0.39	float where an int should be	cast as int	refine_input	consumer	consumerid
4	UNEXPECTED-6i	0.39	float where an int should be	cast as int	refine_input	consumer	consumerid
5	UNEXPECTED-6i	0.39	float where an int should be	cast as int	refine_input	consumer	consumerid
6	UNEXPECTED-6i	0.39	float where an int should be	cast as int	refine_input	consumer	consumerid
7	UNEXPECTED-6i	0.39	float where an int should be	cast as int	refine_input	consumer	consumerid
8	UNEXPECTED-6i	0.39	float where an int should be	cast as int	refine_input	consumer	Age
9	UNEXPECTED-6i	0.39	float where an int should be	cast as int	refine_input	consumer	Age
10	UNEXPECTED-6i	0.39	float where an int should be	cast as int	refine_input	consumer	Age
11	UNEXPECTED-6i	0.39	float where an int should be	cast as int	refine_input	consumer	Age
12	UNEXPECTED-6i	0.39	float where an int should be	cast as int	refine_input	consumer	Age
13	UNEXPECTED-6i	0.39	float where an int should be	cast as int	refine_input	consumer	Age
14	UNEXPECTED-6i	0.39	float where an int should be	cast as int	refine_input	consumer	Age
15	UNEXPECTED-6i	0.39	float where an int should be	cast as int	refine_input	purchase	purchaseid
16	UNEXPECTED-6i	0.39	float where an int should be	cast as int	refine_input	purchase	purchaseid
17	UNEXPECTED-6i	0.39	float where an int should be	cast as int	refine_input	purchase	purchaseid

Markdown Example

	name	severity	description	demo_workaround	pipeline_segment	table_name	field_name
558	BLOCKER-1a	50	PK is not unique in table	mock values	transform	consumer	consumerid
562	BLOCKER-1a	50	PK is not unique in table	mock values	transform	consumer	consumerid
566	BLOCKER-1a	50	PK is not unique in table	mock values	transform	consumer	consumerid
569	BLOCKER-1a	50	PK is not unique in table	mock values	transform	consumer	consumerid
573	BLOCKER-1a	50	PK is not unique in table	mock values	transform	consumer	consumerid
577	BLOCKER-1a	50	PK is not unique in table	mock values	transform	consumer	consumerid
581	BLOCKER-1a	50	PK is not unique in table	mock values	transform	consumer	consumerid
585	BLOCKER-1a	50	PK is not unique in table	mock values	transform	consumer	consumerid
588	BLOCKER-1a	50	PK is not unique in table	mock values	transform	consumer	consumerid
559	MISSING-3a	25	Missing value for a field required for demo	choose random row from real data	transform	consumer	consumerid
563	MISSING-3a	25	Missing value for a field required for demo	choose random row from real data	transform	consumer	consumerid
567	MISSING-3a	25	Missing value for a field required for demo	choose random row from real data	transform	consumer	consumerid
570	MISSING-3a	25	Missing value for a field required for demo	choose random row from real data	transform	consumer	consumerid
574	MISSING-3a	25	Missing value for a field required for demo	choose random row from real data	transform	consumer	consumerid
578	MISSING-3a	25	Missing value for a field required for demo	choose random row from real data	transform	consumer	consumerid
582	MISSING-3a	25	Missing value for a field required for demo	choose random row from real data	transform	consumer	consumerid
586	MISSING-3a	25	Missing value for a field required for demo	choose random row from real data	transform	consumer	consumerid
589	MISSING-3a	25	Missing value for a field required for demo	choose random row from real data	transform	consumer	consumerid
572	RANGE-5a	8	Value too low, out of range	set to minimum in range	transform	consumer	Age
580	RANGE-5a	8	Value too low, out of range	set to minimum in range	transform	consumer	Age
561	RANGE-5b	7.5	Value too high, out of range	set to maximum in range	transform	consumer	Age
565	RANGE-5b	7.5	Value too high, out of range	set to maximum in range	transform	consumer	Age
576	RANGE-5b	7.5	Value too high, out of range	set to maximum in range	transform	consumer	Age
584	RANGE-5b	7.5	Value too high, out of range	set to maximum in range	transform	consumer	Age
557	UNEXPECTED-6b	4.75	Non-ASCII char in str but ASCII output is required	use 'other'	refine_input	fertilizer	type
1	UNEXPECTED-6i	0.39	float where an int should be	cast as int	refine_input	consumer	consumerid
2	UNEXPECTED-6i	0.39	float where an int should be	cast as int	refine_input	consumer	consumerid
3	UNEXPECTED-6i	0.39	float where an int should be	cast as int	refine_input	consumer	consumerid
4	UNEXPECTED-6i	0.39	float where an int should be	cast as int	refine_input	consumer	consumerid
5	UNEXPECTED-6i	0.39	float where an int should be	cast as int	refine_input	consumer	consumerid
6	UNEXPECTED-6i	0.39	float where an int should be	cast as int	refine_input	consumer	consumerid
7	UNEXPECTED-6i	0.39	float where an int should be	cast as int	refine_input	consumer	consumerid
8	UNEXPECTED-6i	0.39	float where an int should be	cast as int	refine_input	consumer	Age
9	UNEXPECTED-6i	0.39	float where an int should be	cast as int	refine_input	consumer	Age
10	UNEXPECTED-6i	0.39	float where an int should be	cast as int	refine_input	consumer	Age
11	UNEXPECTED-6i	0.39	float where an int should be	cast as int	refine_input	consumer	Age
12	UNEXPECTED-6i	0.39	float where an int should be	cast as int	refine_input	consumer	Age
13	UNEXPECTED-6i	0.39	float where an int should be	cast as int	refine_input	consumer	Age
14	UNEXPECTED-6i	0.39	float where an int should be	cast as int	refine_input	consumer	Age
15	UNEXPECTED-6i	0.39	float where an int should be	cast as int	refine_input	purchase	purchaseid
16	UNEXPECTED-6i	0.39	float where an int should be	cast as int	refine_input	purchase	purchaseid
17	UNEXPECTED-6i	0.39	float where an int should be	cast as int	refine_input	purchase	purchaseid

AWS S3 Buckets

Using the local file system as if folders represent buckets, see below. (Not shown: we must write each different customer's output to different locations to protect against write collisions and privacy leaks.)

1. Pipeline Output is in **result**
2. Data Quality Output in **report**