University of Westminster

School of Computer Science and Engineering

Advanced Big Data Analytics (IIT Sri Lanka)

7BDIN004C

Coursework

Analyzing US Restaurant Inspections

Name	:	M.M.C.A. Marasinghe
UOW ID	:	20562027
IIT ID	:	20231760

Table of Contents

Introduction

The food industry, a billion-dollar sector, is essential for humanity and is becoming increasingly important as the global population rises. The industry encompasses a diverse array of stakeholders, connecting everyone on the planet in some way. People have multiple avenues to consume food: they can buy ingredients from retail stores to prepare meals themselves, or they can purchase prepared food from restaurants. These activities generate vast amounts of data every day.

In today's industry, food security is an increasing concern. However, understanding food safety remains challenging. This study aims to analyze datasets related to food safety using tools from the Apache Hadoop stack.

Problem Identification and Literature Review

Research indicates that foodborne illnesses affect 48 million people (about twice the population of Texas) annually in the United States, roughly twice the population of Texas. A significant portion of these illnesses originate from food prepared and served in restaurants. To ensure food safety, local authorities conduct regular inspections of these establishments. The results of these inspections are published on relevant Open Data portals managed by the local authorities.

Open Data portal

An open data portal in America provides public access to various datasets made available by government agencies, non-profit organizations, and other institutions. These portals are part of the broader open data movement, which aims to increase transparency, encourage civic engagement, and foster innovation by making data freely accessible to the public. These datasets are frequently updated by those institutions.

Key features and purposes of open data portals include accessibility, transparency, civic engagement, and innovation. The largest portal, data.gov, serves as the primary source for U.S. federal government data, hosting a diverse array of datasets from numerous federal agencies. In addition to data.gov, many states and cities have their own portals, such as NYC Open Data, the Chicago Data Portal, and California Open Data.

Problem

Although there are large amounts of data, those data are not uniform across states and cities which makes analyzing these large amounts of data difficult. Different states have their own schemas. The number of columns also differs from state to state. Categorizing methods are also changing from state to state. As there is not any unified way to record these data, there is no way to connect illnesses to the violations. There is no way to validate, restaurant inspection methodology.

On top of this complexity, there comes the other traditional problems associated with big data. When all the datasets of states and cities are combined, this dataset is going to be a very large dataset. As I previously mentioned, these data are updated by respective authorities frequently which creates large amounts of data.

Solution

To address the above-mentioned issue, a big data analytics tool using Apache Hadoop stack to analyze these datasets. Along with this tool common schema was developed analyzing these datasets.

Literature Review

Restaurant inspection data is recorded in two methods. One is giving a score based on the results of inspections. Another one is recording the type of violation with a status. From a study done in 2004, it was found that mean scores of restaurants experiencing foodborne disease outbreaks did not differ from restaurants with no reported outbreaks. This questions the validity of recording a score based the on-restaurant inspection. The same conclusion can be found in another study too.

For this study, researchers have used data from Tennessee from January 1993 through April 2000. During this time, there were 167574 inspections involving 29008 restaurants and 248 inspectors. They have not used big data analytics tools as they have studied only a small dataset.

According to the study, the most common violations are “Nonfood contact surfaces of equipment and utensils clean” and “Floors constructed, drained, clean, good repair, covering, installation, dustless cleaning methods”. None of the twelve most frequently cited violations were classified as "critical" food safety hazards. The most cited critical violation was the “improper storage or use of toxic items, such as storing cleaning fluids on a shelf next to food”.

Inspection data were available from 49 restaurants identified as the source of foodborne disease outbreaks investigated by health departments in Tennessee from 1999 to 2002. The average score of the last routine inspection before the reported outbreak was 81.2, and the average score before the most recent one was 81.6. These scores do not significantly differ from the average scores of all restaurant inspections during the study period. This tells us that more deep analysis is needed.

In another study, NYC Data Science Academy has done analysis on New York Restaurant inspections. They have not used advanced data engineering tools for this. They have used MariaDB for storage, Python for processing and Pandas and PyPlot for data visualization. Similar data analysis, can be found for Los Angeles Environmental Health Restaurant and Market Violations dataset. Similar data analysis can be found for San Fransisco Restaurant inspection dataset too.

Most recent data standard can be found here. In 2012, Yelp collaborated with the cities of San Francisco and New York to create the Local Inspector Value-Entry Specification (LIVES). LIVES is an open data standard that enables municipalities to publish restaurant inspection information on Yelp. Now multiple government bodies have joined this program. They have created multiple tables businesses, inspections, feed_info, violations and legend which is superior to previous single table standard. This standard is much more scalable and extendable.

When studying this domain, I could not find a big data related analytics solution for this problem.

Data collection and management

Datasets

I used three datasets for the analysis. These datasets are available in csv format.

1. King Country Food establishment inspection data
2. Chicago Food inspection data
3. NYC DOHMH New York City Restaurant Inspection Results

King Country food establishment inspection data

This dataset has 22 columns.

Column Name	Description	API Field Name	Data Type
Name		name	Text
Program Identifier		program_identifier	Text
Inspection Date		inspection_date	Floating Timestamp
Description		description	Text
Address		address	Text
City		city	Text
Zip Code		zip_code	Text
Phone		phone	Text
Longitude		longitude	Number
Latitude		latitude	Number
Inspection Business Name		inspection_business_name	Text
Inspection Type		inspection_type	Text
Inspection Score		inspection_score	Number
Inspection Result		inspection_result	Text
Inspection Closed Business		inspection_closed_business	Checkbox
Violation Type		violation_type	Text
Violation Description		violation_description	Text
Violation Points		violation_points	Number
Business_ID		business_id	Text
Inspection_Serial_Num		inspection_serial_num	Text
Violation_Record_ID		violation_record_id	Text
Grade	Food establishment grade	grade	Text

Chicago Food Inspection data

This dataset has 17 columns.

Column Name	Description	API Field Name	Data Type
Inspection ID		inspection_id	Number
DBA Name	Doing Business As	dba_name	Text
AKA Name	Also Known As	aka_name	Text
License #		license_	Number
Facility Type		facility_type	Text
Risk		risk	Text
Address		address	Text
City		city	Text
State		state	Text
Zip		zip	Number
Inspection Date		inspection_date	Floating Timestamp
Inspection Type		inspection_type	Text
Results		results	Text
Violations		violations	Text
Latitude		latitude	Number

NYC DOHMH New York City Restaurant Inspection Results

This dataset has 27 columns.

Column Name	Description	API Field Name	Data Type
CAMIS	This is an unique identifier for the entity (restaurant); 10-digit integer, static per restaurant permit	camis	Text
DBA	This field represents the name (doing business as) of the entity (restaurant); Public business name, may change at discretion of restaurant owner	dba	Text
BORO	Borough in which the entity (restaurant) is located.;• 1 = MANHATTAN • 2 = BRONX • 3 = BROOKLYN • 4 = QUEENS • 5 = STATEN ISLAND • Missing; NOTE: There may be... Read more	boro	Text
BUILDING	Building number for establishment (restaurant) location	building	Text
STREET	Street name for establishment (restaurant) location	street	Text
ZIPCODE	Zip code of establishment (restaurant) location	zipcode	Text
PHONE	Phone Number; Phone number provided by restaurant owner/manager	phone	Text
CUISINE DESCRIPTION	This field describes the entity (restaurant) cuisine. ; Optional field provided by provided by restaurant owner/manager	cuisine_description	Text
INSPECTION DATE	This field represents the date of inspection; NOTE: Inspection dates of 1/1/1900 mean an establishment has not yet had an inspection	inspection_date	Floating Timestamp
ACTION	This field represents the actions that is associated with each restaurant inspection. ; • Violations were cited in the following area(s). • No violations were recorded at the time... Read more	action	Text
VIOLATION CODE	Violation code associated with an establishment (restaurant) inspection	violation_code	Text
VIOLATION DESCRIPTION	Violation description associated with an establishment (restaurant) inspection	violation_description	Text
CRITICAL FLAG	Indicator of critical violation; "• Critical • Not Critical • Not Applicable"; Critical violations are those most likely to contribute to food-borne illness	critical_flag	Text
SCORE	Total score for a particular inspection; Scores are updated based on adjudication results	score	Number
GRADE	Grade associated with the inspection; • N = Not Yet Graded• A = Grade A• B = Grade B• C = Grade C• Z = Grade Pending• P= Grade Pending issued on re-opening following an initial... Read more	grade	Text

Column Name	Description	API Field Name	Data Type
GRADE DATE	The date when the current grade was issued to the entity (restaurant)	grade_date	Floating Timestamp
RECORD DATE	The date when the extract was run to produce this data set	record_date	Floating Timestamp
INSPECTION TYPE	A combination of the inspection program and the type of inspection performed; See Data Dictionary for full list of expected values	inspection_type	Text
Latitude		latitude	Number
Longitude		longitude	Number
Community Board		community_board	Text
Council District		council_district	Text
Census Tract		census_tract	Text
BIN		bin	Text
BBL		bbl	Text
NTA		nta	Text
Location Point1		location_point1	Point

Future Work

There are numerous datasets that can be analyzed using the same approach. Here, I have attached a few examples.

1. https://data.sfgov.org/Health-and-Social-Services/-Historical-Restaurant-Inspection-Scores-2016-2019/pyih-qa8i/about_data
2. https://data.sonomacounty.ca.gov/Health/Food-Facility-Inspections/8r44-w5qd/data

Data Management Policy

Here, we have selected three datasets in the initial version. But the solution we present is extendable to all the inspection datasets. Each row represents an inspection, and if a specific business receives multiple violations during an inspection, there will be multiple rows for that business, all sharing the same **Inspection Identifier**.

As I explained in the problem statement, these datasets have different kinds of schemas. First, we need to create a common schema to analyze.

Universal column name	Description
inspection_date	
restaurant_id	

restaurant_name	
inspection_type	
score	
description	

These columns were identified as the most important columns for analysis. Mapping of each dataset to the above columns can be found in the following tables.

King country

Column name	**Universal column name**
Inspection Date	inspection_date
Program Identifier	restaurant_id
Name	restaurant_name
Inspection Type	inspection_type
Inspection Score	score
Violation Description	description

Chicago

Column name	**Universal column name**
Inspection Date	inspection_date
License #	restaurant_id
DBA Name	restaurant_name
Inspection Type	inspection_type
Risk	score
Violations	description

*Risk should be mapped to a numerical value

NYC

Column name	**Universal column name**
INSPECTION DATE	inspection_date
CAMIS	restaurant_id
DBA	restaurant_name
INSPECTION TYPE	inspection_type
SCORE	score
VIOLATION DESCRIPTION	description

After applying the mapping to these datasets, these datasets will be stored in csv format for further analysis.

Data Processing and Analytics Solution

It has been decided to use the Hadoop big data stack for this analysis. AWS offers a managed Hadoop cluster called Elastic MapReduce (EMR), which simplifies the time-consuming and complex process of configuring a Hadoop cluster. EMR also enables the use of tools such as Spark, Pig, and Hive.

Elastic Map Reduce (EMR)

Dependency	Version
EMR	7.2.0
Livy	0.8.0
Spark	3.5.1
Hadoop	3.3.6
Jupyter Hub	1.5.0
Hive	3.1.3

EMR cluster has three types of nodes

1. **Master** – Cluster coordination
2. **Core** – Processing and data storing
3. **Task** – Processing

Cluster configuration used

Node	Count
Master	1
Core	2
Task	0

EMR cluster was configured to use S3 as input and output storage. When using EMR, there are two approaches for storage configuration.

1. Just using S3
2. Using Hadoop & S3

When working with a very large dataset and planning to apply transformations using Spark and Amazon EMR, deciding between using Hadoop (HDFS) or just S3 depends on several factors.

Using S3

1. **Cost-Effectiveness**: S3 is generally cheaper for storage compared to HDFS, especially for large datasets.
2. **Scalability**: S3 scales automatically to handle large amounts of data, so you don't need to manage the storage infrastructure.

3. **Ease of Use**: S3 is simpler to set up and use, and it integrates well with many AWS services.
4. **Durability and Availability**: S3 provides high durability and availability of data, which is critical for large datasets.
5. **Data Access Patterns**: If you need to access the data frequently or from multiple locations, S3 is more suited because it's designed for high availability and global accessibility.

Using HDFS

1. **Performance**: HDFS can offer better performance for certain workloads, especially if you have a need for high-throughput access to data during processing.
2. **Data Locality**: HDFS benefits from data locality, meaning computation occurs near the data, which can reduce latency and improve processing speed for some workloads.
3. **Complex Workloads**: If your Spark jobs involve complex, iterative processing or require low-latency access to data, HDFS might be more suitable.
4. **Existing Hadoop Ecosystem**: If you already have a Hadoop ecosystem in place and are leveraging other Hadoop tools, it might be easier to integrate with HDFS.

Solution

Due to the cost effectiveness and not having any performance bottlenecks, I have used just S3 with EMR cluster. If there are performance bottlenecks in future, we can go with HDFS. I used JuPyter Hub and JuPyter Notebook for development.

Other Solutions

We can use Pig or Hive for this analytics solution. But I decided to go with Spark as I am very familiar with PySpark.

Data Processing

Dataset 1 - King country

Dataset 2 - Chicago

Dataset 3 - NYC

Initializing Spark

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import col

spark = SparkSession.builder.appName("FirstApp").getOrCreate()

data_url1 = 's3://emr-project/input/Food_Establishment_Inspection_Data_20240724.csv'
data_url2 = 's3://emr-project/input/Food_Inspections_20240726.csv'
data_url3 = 's3://emr-project/input/DOHMH_New_York_City_Restaurant_Inspection_Results_20240725.csv'
output_url = 's3://emr-project/output/'
```

Preprocessing Dataset

During preprocessing the dataset was converted into the common format and saved in S3 file and csv. When saving datasets can be partitioned due to the distributed nature of the processing. But that can be concatenated easily when analyzing.

```
df2 = spark.read.option("header", "true").csv(data_url2)
df2 = df2.select("Inspection Date", "License #", "DBA Name", "Inspection Type", "Risk", "Violations")

df2 = df2.withColumnRenamed("Inspection Date", "inspection_date") \
    .withColumnRenamed("License #", "restaurant_id") \
    .withColumnRenamed("DBA Name", "restaurant_name") \
    .withColumnRenamed("Inspection Type", "inspection_type") \
    .withColumnRenamed("Risk", "score") \
    .withColumnRenamed("Violations", "description")

df2.createOrReplaceTempView("data_set2")

df2.write.mode("overwrite").option("header", "true").csv(output_url + "/dataset2")
```

```
df3 = spark.read.option("header", "true").csv(data_url3)
df3 = df3.select("INSPECTION DATE", "CAMIS", "DBA", "INSPECTION TYPE", "SCORE", "VIOLATION DESCRIPTION")

df3 = df3.withColumnRenamed("INSPECTION DATE", "inspection_date") \
    .withColumnRenamed("CAMIS", "restaurant_id") \
    .withColumnRenamed("DBA", "restaurant_name") \
    .withColumnRenamed("INSPECTION TYPE", "inspection_type") \
    .withColumnRenamed("SCORE", "score") \
    .withColumnRenamed("VIOLATION DESCRIPTION", "description")

df3.createOrReplaceTempView("data_set3")

df3.write.mode("overwrite").option("header", "true").csv(output_url + "/dataset3")
```

```
# Read CSV file from S3
df1 = spark.read.option("header", "true").csv(data_url1)
df1 = df1.select("Inspection Date", "Program Identifier", "Name", "Inspection Type", "Inspection Score", "Violation

df1 = df1.withColumnRenamed("Inspection Date", "inspection_date") \
    .withColumnRenamed("Program Identifier", "restaurant_id") \
    .withColumnRenamed("Name", "restaurant_name") \
    .withColumnRenamed("Inspection Type", "inspection_type") \
    .withColumnRenamed("Inspection Score", "score") \
    .withColumnRenamed("Violation Description", "description")

df1.createOrReplaceTempView("data_set1")

df1.write.mode("overwrite").option("header", "true").csv(output_url + "/dataset1")
```

Analysis and Results

Normalizing

When reading csv, multiple csv files were combined to create one dataframe. Before analyzing datasets should be normalized for uniformity. There was no need to normalize Dataset 2.

```
# Read all CSV files into a single DataFrame
df1 = spark.read.csv("s3://emr-project/output/dataset1/" + "*.csv", header=True, inferSchema=True)

df1 = df1.withColumn("score", col("score").cast("double"))
df1 = df1.filter(df1.score.isNotNull())
df1 = df1.filter(df1.restaurant_id.isNotNull())

min_score, max_score = df1.select(spark_min(col("score")), spark_max(col("score"))).first()

df1 = df1.withColumn("normalized_score", (col("score") - min_score) / (max_score - min_score))
df1 = df1.withColumn("normalized_score", round(col("normalized_score") * 100, 3))

df1 = df1.drop("score")

df1.show()
```

```
# Read all CSV files into a single DataFrame
df2 = spark.read.csv("s3://emr-project/output/dataset2/" + "*.csv", header=True, inferSchema=True)

df2 = df2.withColumn("score", col("score").cast("double"))
df2 = df2.filter(df2.score.isNotNull())

df2 = df2.withColumnRenamed("score", "normalized_score")

df2.show()
```

```
# Read all CSV files into a single DataFrame
df3 = spark.read.csv("s3://emr-project/output/dataset3/" + "*.csv", header=True, inferSchema=True)

df3 = df3.withColumn("score", col("score").cast("double"))
df3 = df3.filter(df3.score.isNotNull())

min_score, max_score = df3.select(spark_min(col("score")), spark_max(col("score"))).first()

df3 = df3.withColumn("normalized_score", (col("score") - min_score) / (max_score - min_score))
df3 = df3.withColumn("normalized_score", round(col("normalized_score") * 100, 3))

df3 = df3.drop("score")

df3.show()
```

Most frequently inspected Restaurants

```
count_df1 = df1.groupBy("restaurant_id").count()
result_df1 = count_df1.join(
    df1.select("restaurant_id", "restaurant_name").distinct(),
    on="restaurant_id",
    how="left"
)
sorted_df1 = result_df1.orderBy(col("count").desc()).limit(10)
sorted_df1.show()
```

restaurant_id	count	restaurant_name
TACO TIME	1127	TACO TIME
SUBWAY	1026	SUBWAY #3146
SUBWAY	1026	SUBWAY
SUBWAY	1026	SUBWAY #10558
SUBWAY	1026	MAPLE VALLEY SUBW...
PAGLIACCI PIZZA INC	687	PAGLIACCI PIZZA INC
TOSHI'S TERIYAKI	412	TOSHI'S TERIYAKI
TAQUERIA EL RINCO...	408	TAQUERIA EL RINCO...
CAFFE LADRO	397	CAFFE LADRO
MCDONALD'S	333	MCDONALD'S #11027

```
aggregated_names_df2 = df2
    .groupBy("restaurant_id")
    .agg(concat_ws(", ", collect_set("restaurant_name")).alias("restaurant_names"))
count_df2 = df2.groupBy("restaurant_id").count()
result_df2 = count_df2.join(aggregated_names_df2, on="restaurant_id", how="left")
sorted_df2 = result_df2.orderBy(col("count").desc()).limit(10)

sorted_df2.show()
```

```
+-------------+-----+--------------------+
|restaurant_id|count|    restaurant_names|
+-------------+-----+--------------------+
|            0|  673|EPIPHANY PARISH/C...|
|      1354323|  198|SPORTS SERVICE SO...|
|        14616|  183|ILLINOIS SPORTSER...|
|      1574001|   87|LEVY RESTAURANTS ...|
|        60184|   64|TAQUERIA EL RANCHITO|
|      2083833|   62|MARIANO'S FRESH M...|
|      1142451|   60|JEWEL FOOD  STORE...|
|      1974745|   59|LEVY RESTAURANTS ...|
|        25152|   56|Chavez Upper Grad...|
|      1884255|   56|FOOD 4 LESS, FOOD...|
+-------------+-----+--------------------+
```

```
count_df3 = df3.groupBy("restaurant_id").count()
result_df3 = count_df3.join(
    df3.select("restaurant_id", "restaurant_name").distinct(),
    on="restaurant_id",
    how="left"
)
sorted_df3 = result_df3.orderBy(col("count").desc()).limit(10)
sorted_df3.show()
```

```
+-------------+-----+--------------------+
|restaurant_id|count|     restaurant_name|
+-------------+-----+--------------------+
|     40365904|   64|        MEE SUM CAFE|
|     50111296|   63| BIG WONG RESTAURANT|
|     40398688|   60|          MASTER WOK|
|     41406895|   59|SUN SAI GAI RESTA...|
|     50089474|   59|    THE COPPOLA CAFE|
|     50105561|   55|   DAVIDOVICH BAKERY|
|     50079599|   52|      NICE ONE BAKERY|
|     50123073|   52|    PI GREEK BAKERIE|
|     50044250|   47|FUJI JAPANESE CUI...|
|     41658324|   46|  MI CASA RESTAURANT|
+-------------+-----+--------------------+
```

Most Common Inspection Type

Authorities can get an idea whether we should increase routine inspections or not.

```
count_df1 = df1.groupBy("inspection_type").count()
sorted_df1 = count_df1.orderBy(col("count").desc()).limit(10)
sorted_df1.show()
```

```
+--------------------+------+
|     inspection_type| count|
+--------------------+------+
|Routine Inspectio...|218396|
|Consultation/Educ...| 38718|
|   Return Inspection|  6764|
+--------------------+------+
```

```
count_df3 = df3.groupBy("inspection_type").count()
sorted_df3 = count_df3.orderBy(col("count").desc()).limit(10)
sorted_df3.show()
```

```
+--------------------+------+
|     inspection_type| count|
+--------------------+------+
|Cycle Inspection ...|132395|
|Cycle Inspection ...| 41967|
|Pre-permit (Opera...| 34625|
|Pre-permit (Opera...|  9883|
|Pre-permit (Non-o...|  3092|
|Pre-permit (Opera...|  1648|
|Cycle Inspection ...|  1442|
|Cycle Inspection ...|   968|
|Pre-permit (Opera...|   733|
|Inter-Agency Task...|   381|
+--------------------+------+
```

```
count_df2 = df2.groupBy("inspection_type").count()
sorted_df2 = count_df2.orderBy(col("count").desc()).limit(10)
sorted_df2.show()
```

```
+--------------------+------+
|     inspection_type| count|
+--------------------+------+
|             Canvass|143364|
|             License| 36772|
|Canvass Re-Inspec...| 30598|
|           Complaint| 26020|
|License Re-Inspec...| 11706|
|Complaint Re-Insp...| 10919|
|Short Form Complaint|  8440|
|      Non-Inspection|  3248|
|Suspected Food Po...|   973|
|        Consultation|   674|
+--------------------+------+
```

Inspection records that are older than two years

```
# Convert the Inspection_date column to date type
df1 = df1.withColumn("inspection_date", to_date(col("inspection_date"), "MM/dd/yyyy"))

# Calculate the date 2 years ago from today
two_years_ago = date_sub(current_date(), 2 * 365)

old_inspections_df1 = df1.filter(col("inspection_date") < two_years_ago)
old_inspections_df1 = old_inspections_df1.drop("restaurant_id")

# Show the filtered DataFrame
old_inspections_df1.show()
```

```
+---------------+----------------+--------------------+--------------------+----------------+
|inspection_date| restaurant_name|     inspection_type|         description|normalized_score|
+---------------+----------------+--------------------+--------------------+----------------+
|     2022-01-13|#807 TUTTA BELLA|Routine Inspectio...|                NULL|           5.319|
|     2021-01-06|#807 TUTTA BELLA|Routine Inspectio...|                NULL|           5.319|
|     2022-07-13|       +MAS CAFE|    Return Inspection|                NULL|           5.319|
|     2022-06-29|       +MAS CAFE|Routine Inspectio...|3400 - Wiping clo...|          30.319|
|     2022-06-29|       +MAS CAFE|Routine Inspectio...|4800 - Physical f...|          30.319|
|     2022-06-29|       +MAS CAFE|Routine Inspectio...|3200 - Insects, r...|          30.319|
|     2022-06-29|       +MAS CAFE|Routine Inspectio...|1600 - Proper coo...|          30.319|
|     2022-06-29|       +MAS CAFE|Routine Inspectio...|2110 - Proper col...|          30.319|
|     2021-12-29|       +MAS CAFE|Routine Inspectio...|                NULL|           5.319|
|     2020-07-29|       +MAS CAFE|Consultation/Educ...|                NULL|           5.319|
|     2022-07-13|      100 LB CLAM|Consultation/Educ...|                NULL|           5.319|
|     2019-09-12|      100 LB CLAM|Routine Inspectio...|                NULL|           5.319|
|     2017-07-24|      100 LB CLAM|Routine Inspectio...|3300 - Potential ...|          18.617|
|     2017-07-24|      100 LB CLAM|Routine Inspectio...|0200 - Food Worke...|          18.617|
|     2017-07-24|      100 LB CLAM|Routine Inspectio...|0600 - Adequate h...|          18.617|
|     2017-07-24|      100 LB CLAM|Routine Inspectio...|2300 - Proper Con...|          18.617|
|     2017-07-06|      100 LB CLAM|Consultation/Educ...|                NULL|           5.319|
|     2016-08-31|      100 LB CLAM|Consultation/Educ...|                NULL|           5.319|
|     2022-06-03|  100TH AVE CAKES|Consultation/Educ...|                NULL|           5.319|
|     2022-04-29|  100TH AVE CAKES|Routine Inspectio...|                NULL|           5.319|
+---------------+----------------+--------------------+--------------------+----------------+
only showing top 20 rows
```

```
# Convert the Inspection_date column to date type
df2 = df2.withColumn("inspection_date", to_date(col("inspection_date"), "MM/dd/yyyy"))

# Calculate the date 2 years ago from today
two_years_ago = date_sub(current_date(), 2 * 365)

old_inspections_df2 = df2.filter(col("inspection_date") < two_years_ago)
old_inspections_df2 = old_inspections_df2.drop("restaurant_id")

# Show the filtered DataFrame
old_inspections_df2.show()
```

```
+---------------+--------------------+--------------------+----------------+--------------------+
|inspection_date|     restaurant_name|     inspection_type|normalized_score|         description|
+---------------+--------------------+--------------------+----------------+--------------------+
|     2018-06-08|        JET'S PIZZA|Canvass Re-Inspec...|           33.33|45. FOOD HANDLER ...|
|     2018-06-12|SIMPLY SOUPS SALA...|             Canvass|           33.33|2. FACILITIES TO ...|
|     2018-06-29|COLUMBUS MANOR RE...|Canvass Re-Inspec...|           33.33|33. FOOD AND NON-...|
|     2018-07-24|MEZZA GRILLED WRA...|             Canvass|           33.33|3. MANAGEMENT, FO...|
|     2018-08-14|GATELY SEAFOOD MA...|             License|           33.33|                NULL|
|     2018-08-07|BISMILLAH RESTAURANT|Canvass Re-Inspec...|           33.33|58. ALLERGEN TRAI...|
|     2018-06-05|     FRANCESCA'S CAFE|License Re-Inspec...|           33.33|                NULL|
|     2018-06-21|PRECIOUS LITTLE O...|             License|           33.33|35. WALLS, CEILIN...|
|     2018-07-09|PATRON MEXICAN GR...|           Complaint|           33.33|3. MANAGEMENT, FO...|
|     2018-08-03|              SUBWAY|             Canvass|           33.33|3. MANAGEMENT, FO...|
|     2018-06-20|              ALDI'S|             Canvass|           100.0|34. FLOORS: CONST...|
|     2018-07-18|JEWEL FOOD STORE ...|Short Form Complaint|           33.33|                NULL|
|     2018-06-01|          VISTA CAFE|             Canvass|           33.33|11. ADEQUATE NUMB...|
|     2018-07-26|    LONDONHOUSE HOTEL|             License|           100.0|                NULL|
|     2018-06-27|LAWNDALE COMMUNIT...|             License|           33.33|34. FLOORS: CONST...|
|     2018-06-20|          BIG SHARKS|             License|           33.33|                NULL|
|     2018-06-04|         TY FOOD MART|             License|           100.0|                NULL|
|     2018-07-26|         ROSEBUD-RUSH|           Complaint|           33.33|3. MANAGEMENT, FO...|
|     2018-07-11|SPRINKLES CUPCAKE...|             Canvass|           66.66|3. MANAGEMENT, FO...|
|     2018-06-08|AFC SUSHI@JEWEL O...|             Canvass|           33.33|                NULL|
+---------------+--------------------+--------------------+----------------+--------------------+
only showing top 20 rows
```

```
# Convert the Inspection_date column to date type
df3 = df3.withColumn("inspection_date", to_date(col("inspection_date"), "MM/dd/yyyy"))

# Calculate the date 2 years ago from today
two_years_ago = date_sub(current_date(), 2 * 365)

old_inspections_df3 = df3.filter(col("inspection_date") < two_years_ago)
old_inspections_df3 = old_inspections_df3.drop("restaurant_id")

# Show the filtered DataFrame
old_inspections_df2.show()
```

```
+---------------+--------------------+--------------------+----------------+--------------------+
|inspection_date|     restaurant_name|     inspection_type|normalized_score|         description|
+---------------+--------------------+--------------------+----------------+--------------------+
|     2018-06-08|        JET'S PIZZA|Canvass Re-Inspec...|           33.33|45. FOOD HANDLER ...|
|     2018-06-12|SIMPLY SOUPS SALA...|             Canvass|           33.33|2. FACILITIES TO ...|
|     2018-06-29|COLUMBUS MANOR RE...|Canvass Re-Inspec...|           33.33|33. FOOD AND NON-...|
|     2018-07-24|MEZZA GRILLED WRA...|             Canvass|           33.33|3. MANAGEMENT, FO...|
|     2018-08-14|GATELY SEAFOOD MA...|             License|           33.33|                NULL|
|     2018-08-07|BISMILLAH RESTAURANT|Canvass Re-Inspec...|           33.33|58. ALLERGEN TRAI...|
|     2018-06-05|     FRANCESCA'S CAFE|License Re-Inspec...|           33.33|                NULL|
|     2018-06-21|PRECIOUS LITTLE O...|             License|           33.33|35. WALLS, CEILIN...|
|     2018-07-09|PATRON MEXICAN GR...|           Complaint|           33.33|3. MANAGEMENT, FO...|
|     2018-08-03|              SUBWAY|             Canvass|           33.33|3. MANAGEMENT, FO...|
|     2018-06-20|              ALDI'S|             Canvass|           100.0|34. FLOORS: CONST...|
|     2018-07-18|JEWEL FOOD STORE ...|Short Form Complaint|           33.33|                NULL|
|     2018-06-01|          VISTA CAFE|             Canvass|           33.33|11. ADEQUATE NUMB...|
|     2018-07-26|    LONDONHOUSE HOTEL|             License|           100.0|                NULL|
|     2018-06-27|LAWNDALE COMMUNIT...|             License|           33.33|34. FLOORS: CONST...|
|     2018-06-20|          BIG SHARKS|             License|           33.33|                NULL|
|     2018-06-04|         TY FOOD MART|             License|           100.0|                NULL|
|     2018-07-26|         ROSEBUD-RUSH|           Complaint|           33.33|3. MANAGEMENT, FO...|
|     2018-07-11|SPRINKLES CUPCAKE...|             Canvass|           66.66|3. MANAGEMENT, FO...|
|     2018-06-08|AFC SUSHI@JEWEL O...|             Canvass|           33.33|                NULL|
+---------------+--------------------+--------------------+----------------+--------------------+
only showing top 20 rows
```

Restaurants that were not inspected in last 2 years

Authorities can use this data to select restaurants that should be inspected soon.

```python
# Convert the Inspection_date column to date type
df1 = df1.withColumn("inspection_date", to_date(col("inspection_date"), "MM/dd/yyyy"))

# Define a window specification to partition by restaurant_id and order by Inspection_date descending
window_spec = Window.partitionBy("restaurant_id").orderBy(desc("inspection_date"))

# Add a row number to each row within the partition
df1_with_row_num = df1.withColumn("row_num", row_number().over(window_spec))

# Filter to get only the latest record for each restaurant_id
latest_records_df1 = df1_with_row_num.filter(col("row_num") == 1).drop("row_num")

# Calculate the date two years ago from today
two_years_ago = date_sub(current_date(), 2 * 365)

# Filter records that are older than two years from latest_records_df
old_records_df = latest_records_df1.filter(col("inspection_date") < two_years_ago)

old_records_df = old_records_df.drop("restaurant_id")

# Show the filtered DataFrame
old_records_df.show()
```

```
+---------------+--------------------+--------------------+--------------------+----------------+
|inspection_date|     restaurant_name|     inspection_type|         description|normalized_score|
+---------------+--------------------+--------------------+--------------------+----------------+
|     2022-03-26|  ADAMS BENCH WINERY|Routine Inspectio...|                NULL|           5.319|
|     2022-03-16|AMAZON - INVENT C...|Routine Inspectio...|                NULL|           5.319|
|     2022-03-16|AMAZON S - MARKET...|Routine Inspectio...|                NULL|           5.319|
|     2022-03-16|    AMAZON S- KITCHEN|Routine Inspectio...|                NULL|           5.319|
|     2022-04-02|AMBASSADOR WINES ...|Routine Inspectio...|                NULL|           5.319|
|     2019-03-12|AMERICAN LEGION P...|Routine Inspectio...|                NULL|           5.319|
|     2022-05-21|ANCESTRY CELLARS ...|Routine Inspectio...|0600 - Adequate h...|          10.638|
|     2022-03-26|ANCESTRY CELLARS,...|Routine Inspectio...|                NULL|           5.319|
|     2022-03-24|ANCHORHEAD COFFEE...|Routine Inspectio...|                NULL|           5.319|
|     2021-07-15|ANTOJITOS LA DONA...|Consultation/Educ...|                NULL|           5.319|
|     2022-04-15|ARMSTRONG FAMILY ...|Routine Inspectio...|                NULL|           5.319|
|     2019-04-05|AVANTI MARKETS NW...|Routine Inspectio...|                NULL|           5.319|
|     2019-04-05|AVANTI MARKETS NW...|Routine Inspectio...|                NULL|           5.319|
|     2022-03-26|Ambassador Wines ...|Routine Inspectio...|                NULL|           5.319|
|     2022-04-02|BAER WINERY - TAS...|Routine Inspectio...|                NULL|           5.319|
|     2022-03-26|BAER WINERY WOODI...|Routine Inspectio...|                NULL|           5.319|
|     2022-04-02|BARRAGE CELLARS -...|Routine Inspectio...|0600 - Adequate h...|          10.638|
|     2022-04-14|BEAUMONT CELLARS ...|Routine Inspectio...|                NULL|           5.319|
|     2019-06-20|       BEST CATERING|Routine Inspectio...|                NULL|           5.319|
|     2019-08-28|BEST OF BOTH WORL...|Routine Inspectio...|                NULL|           5.319|
+---------------+--------------------+--------------------+--------------------+----------------+
only showing top 20 rows
```

```python
# Convert the Inspection_date column to date type
df2 = df2.withColumn("inspection_date", to_date(col("inspection_date"), "MM/dd/yyyy"))

# Define a window specification to partition by restaurant_id and order by Inspection_date descending
window_spec = Window.partitionBy("restaurant_id").orderBy(desc("inspection_date"))

# Add a row number to each row within the partition
df2_with_row_num = df2.withColumn("row_num", row_number().over(window_spec))

# Filter to get only the latest record for each restaurant_id
latest_records_df2 = df2_with_row_num.filter(col("row_num") == 1).drop("row_num")

# Calculate the date two years ago from today
two_years_ago = date_sub(current_date(), 2 * 365)

# Filter records that are older than two years from latest_records_df
old_records_df = latest_records_df2.filter(col("inspection_date") < two_years_ago)

old_records_df = old_records_df.drop("restaurant_id")

# Show the filtered DataFrame
old_records_df.show()
```

inspection_date	restaurant_name	inspection_type	normalized_score	description
2017-02-24	CAL'S 400 LIQUORS...	Canvass	100.0	NULL
2012-07-25	PIZZA LOUNGE	Canvass	100.0	NULL
2012-10-04	FRANCES COCKTAIL ...	Canvass	100.0	NULL
2011-09-08	LLOYDS LOUNGE, INC.	Complaint Re-Insp...	100.0	33. FOOD AND NON-...
2019-04-29	TREASURE ISLAND F...	Canvass	33.33	NULL
2019-06-28	TREASURE ISLAND F...	Canvass	33.33	NULL
2016-11-07	MIRABELL RESTAURANT	Canvass	33.33	NULL
2013-08-20	Orly's/Jalapeno	Canvass	33.33	NULL
2010-03-08	SCHULZE & BURCH B...	Complaint	66.66	33. FOOD AND NON-...
2013-08-21	WHITE STOKES CO INC	Complaint	66.66	9. WATER SOURCE: ...
2022-05-17	GEPPERTH'S MARKET...	Canvass	33.33	NULL
2013-11-14	IRVING PULASKI SH...	Canvass	100.0	NULL
2020-06-25	KUNKA PHARMACY	Canvass	100.0	NULL
2013-03-12	MR SHRIMP	Canvass	66.66	NULL
2019-12-13	ROMANIAN KOSHER S...	Canvass	66.66	3. MANAGEMENT, FO...
2012-05-10	GEORGE TSILIMIGRAS	Canvass	100.0	NULL
2021-06-02	GARFIELD GYROS	Canvass	33.33	NULL
2013-01-29	BEACHVIEW LIQUORS	Canvass	100.0	33. FOOD AND NON-...
2017-03-31	FIVE FACES ICE CR...	Canvass	33.33	33. FOOD AND NON-...
2011-07-18	BUDACKI'S DRIVE INN	Canvass	100.0	NULL

only showing top 20 rows

```python
# Convert the Inspection_date column to date type
df3 = df3.withColumn("inspection_date", to_date(col("inspection_date"), "MM/dd/yyyy"))

# Define a window specification to partition by restaurant_id and order by Inspection_date descending
window_spec = Window.partitionBy("restaurant_id").orderBy(desc("inspection_date"))

# Add a row number to each row within the partition
df3_with_row_num = df3.withColumn("row_num", row_number().over(window_spec))

# Filter to get only the latest record for each restaurant_id
latest_records_df3 = df3_with_row_num.filter(col("row_num") == 1).drop("row_num")

# Calculate the date two years ago from today
two_years_ago = date_sub(current_date(), 2 * 365)

# Filter records that are older than two years from latest_records_df
old_records_df = latest_records_df3.filter(col("inspection_date") < two_years_ago)

old_records_df = old_records_df.drop("restaurant_id")

# Show the filtered DataFrame
old_records_df.show()
```

```
+---------------+--------------------+--------------------+--------------------+----------------+
|inspection_date|     restaurant_name|     inspection_type|         description|normalized_score|
+---------------+--------------------+--------------------+--------------------+----------------+
|     2019-05-22|SOMBA VILLAGE (BA...|Cycle Inspection ...|Cold food item he...|           4.762|
|     2022-03-11|        CAFE CARDINI|Cycle Inspection ...|Food contact surf...|           7.143|
|     2018-10-24|     BROADWAY THEATRE|Cycle Inspection ...|Non-food contact ...|           7.143|
|     2019-06-21|  TOTONNO'S PIZZERIA|Cycle Inspection ...|Food not cooled b...|           4.167|
|     2022-04-26|     ACME BAR & GRILL|Cycle Inspection ...|Raw, cooked or pr...|           7.143|
|     2019-04-30|          RUMPUS ROOM|Cycle Inspection ...|Food contact surf...|           4.167|
|     2018-11-09|RADIO CITY MUSIC ...|Cycle Inspection ...|Filth flies or fo...|           5.952|
|     2022-04-07|HEARST GOOD HOUSE...|Cycle Inspection ...|Non-food contact ...|            1.19|
|     2019-09-12|CITY ISLAND YACHT...|Cycle Inspection ...|Non-food contact ...|           1.786|
|     2022-04-26|UNCLE LOUIE G'S I...|Cycle Inspection ...|Evidence of mice ...|           5.357|
|     2019-05-29|CLUB WYNN - GERA ...|Cycle Inspection ...|Raw, cooked or pr...|           7.738|
|     2019-06-05|CITI FIELD STAND 110|Cycle Inspection ...|Plumbing not prop...|            1.19|
|     2019-06-05|CITI FIELD BALLPA...|Cycle Inspection ...|Non-food contact ...|           5.357|
|     2019-06-05|CITI FIELD NATHAN...|Cycle Inspection ...|Plumbing not prop...|           3.571|
|     2019-08-14|STAND 216B DELTA ...|Cycle Inspection ...|Hot food item not...|           5.357|
|     2019-08-14|           STAND 217|Cycle Inspection ...|Proper sanitizati...|           5.357|
|     2019-09-21|         LEGENDS 000|Cycle Inspection ...|Food not protecte...|           2.976|
|     2020-02-19|APOLLO THEATRE CO...|Cycle Inspection ...|Hand washing faci...|           8.929|
|     2020-03-16|MATAMIM TAKE OUT ...|Cycle Inspection ...|Food not protecte...|           4.167|
|     2019-05-16|          WORLD BEAN|Cycle Inspection ...|Food prepared fro...|           6.548|
+---------------+--------------------+--------------------+--------------------+----------------+
only showing top 20 rows
```

Evaluation and Comparison

When evaluating this we can say this solution is suitable for analysis of these kinds of datasets. The same approach can be followed for all the other datasets in Open Data Inspection datasets. As we have used here, a preprocessing layer we were able to unify all those datasets. Therefore, we were able to apply the same operations in analysis. That simplified the analysis part.

When comparing with other big data processing tools like Pig and Hive, Spark seems to be the easiest one to use. Spark has a library called from which we can run Spark jobs using python scripts. In addition to this, spark solution is extendable for even real time solution which streams data using either Kafka stream or any other method. When doing analytics using Pig and Hive, the learning curve is large, and that solution is also not extendable.

Here I am using S3 as the storage layer in the big data solution. It is cost effective. But when we need more performance, we must use HDFS as the storage layer. We can easily switch to that EMR. Not having sophisticated charts can also be considered a weakness in this solution.

As later developments, we can extend this solution to analyze all those datasets together. It can generate more insightful information for the government.

References

1. https://conservancy.umn.edu/items/fb86e218-b048-4e29-8c37-02e5e3af78b7
2. Jones TF, Pavlin BI, LaFleur BJ, Ingram LA, Schaffner W. Restaurant inspection scores and foodborne disease. Emerg Infect Dis. 2004 Apr;10(4):688-92. doi: 10.3201/eid1004.030343. PMID: 15200861; PMCID: PMC3323064.