Ecosystem Health Monitoring

Python and ELK POC

Victoria Sun- AI Vision Insider

Agenda

- Development environment
- Data loading
- Data parsing,
- Health Metrics
  - Tier 1
  - Tier 2
  - Tier 3

In the scope of Objective, Requirement, issue encountered, Code review

And Conclusion

Objective

Using a csv data file, simulate data parsing and generate ***multi-tier*** metrics to evaluate an ecosystem’s health.

Development Environment

Hardware Intel:

Core I5-1600K LGA1700

ASUS Z790-V Wifi Motherboard

A750GL PCIES Power Supply

GeForceRTX 4060 Ti

Operating System: Windows 10 Home

Data Source: **https://www.kaggle.com/datasets/arshkon/linkedin-job-postings/data**

CSV file generated from web scraper provided by Arsh Kon and collaborators

Software environment:

Elasticsearch ver 8.15.0

Kibana ver 18.15.0

Development environment:

Jupyter lab python kernel

Painless scripting language

Elasticsearch python API

pandas python package

Datetime python package

Arrow python package

About the Data

Public data provided by : https://www.kaggle.com/datasets/arshkon/linkedin-job-postings/data

- Data collected from 123,849 job postings from linkedin.
- Data was collected in the month of April and does not include any previously deleted entries
- Fields of relevance:
  - Post time- this was converted from date time format to string format
  - Location - almost all entries were in the United States, but there was no standardized format for the location
  - Description- describes what the job position entails
  - Formatted experience level - examples include internship, entry, mid-senior, and director
  - Median salary - medial expected salary of the job position (this value could represent the hourly or yearly wage
  - Pay_period- specifies if the median salary listed represents an hourly or yearly wage
  - Title- title of the job position

Data Loading

Objective: Prepare the data to be ready for future metrics generation.

Requirements: load 120k entries via csv file into ELK index

Issue Encountered	**Solution**
When loading large quantities of data into ELK via Jupyter Lab's python IDE, memory and bitrate may be exceeded due to system configurations designed to prevent the system from failing.	Edit the jupyter_server_config.py file where the setting is denoted as c.ServerApp.iopub_data_rate_limit = 100000 (default). Increase the data rate limit. If this file does not exist yet, it can be generated via command line using *jupyter server --generate-config*
Long processing time causes general troubleshooting to be difficult	Use cardinal data and index to test code on smaller dataset first.

Conclusion: Data loading takes over 15 minutes. Data loading code can be found between cells 3 and 47 in the script

Data Parsing

Objective: Prepare the data to be ready for future metrics generation.

Requirements:

- Remove outliers with peculiar salary listings
- Create new field ‘yearly_salary’ based on the existing fields ‘salary’ and ‘pay_period’
- Create new field ‘state’ based on the existing field ‘location’
- Create new field ‘post_month’ based on the existing field ‘original_listing_time’
- Create new field ‘region’ based on the field ‘state’
- Create new field ‘category’ based on the existing field ‘title’

Data Cleaning Anomalies

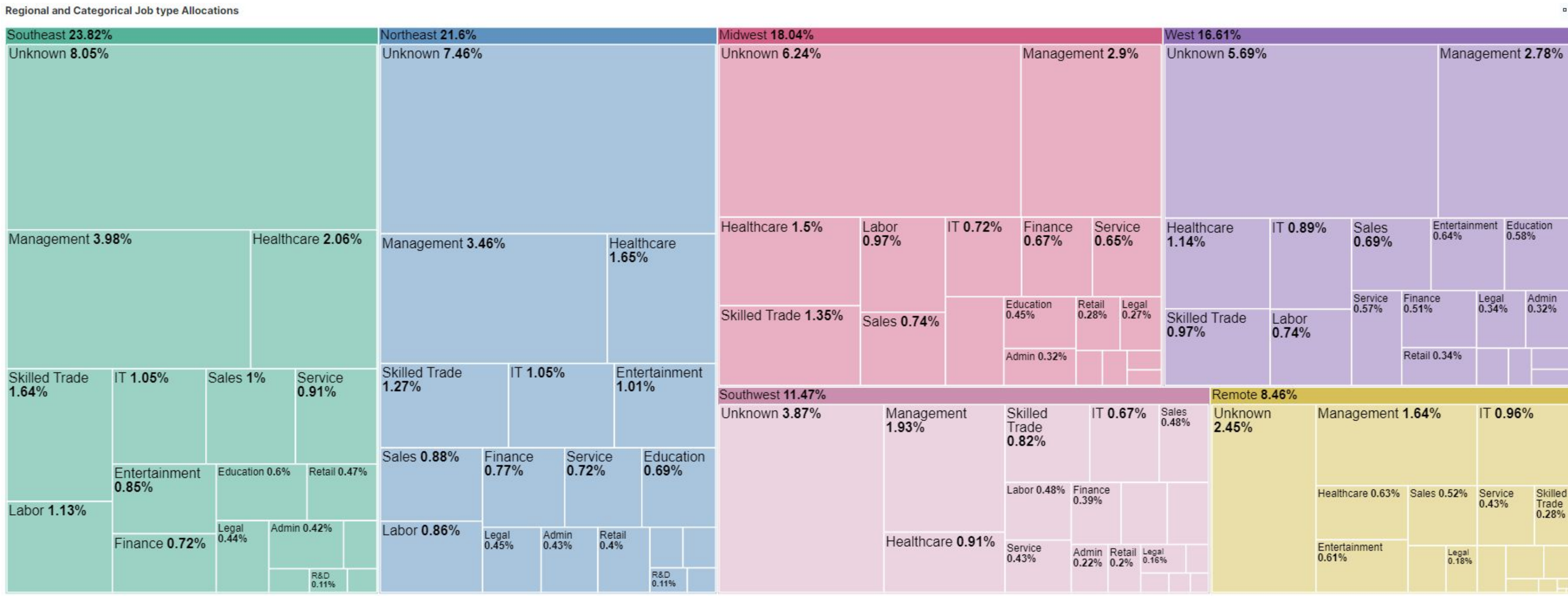
- 2,007 out of 123,859 records were not given a state
- Entries with the location field labeled “United States” were labeled as “Remote” positions
- 47 of the original entries were deemed to have salary listings too abnormal to be considered for further analysis, and were removed from the dataset.

Data Parsing - Results

- Parsing time can exceed 60 seconds based on the query
  - This can potentially be changed through the use of a different software/API instead of python
- Most jobs listing were found in the Southeast region of the United States
- Jobs whose category could not immediately be recognized by the titles were assigned the category ‘Unknown’, this included a significant portion of the data
- Parsing code is found between cells 48 and 140 in the script

Issues Encountered	**Solution**
When loading large quantities of data into ELK via Jupyter Lab’s python IDE or if querying on a large dataset, the error “Connection Timed out” may prevent functions from completing.	When creating an connection to the local elasticsearch instance, establish the timeout interval
Painless scripting returns BadRequestError(400, 'script_exception', 'compile error') When using data arrays	Don’t use painless

Data Parsing - Results



Multilevel Metrics Generation- Tier 1

Objective: Based on Elasticsearch native aggregation function, populate the health index called ‘jobs_health’

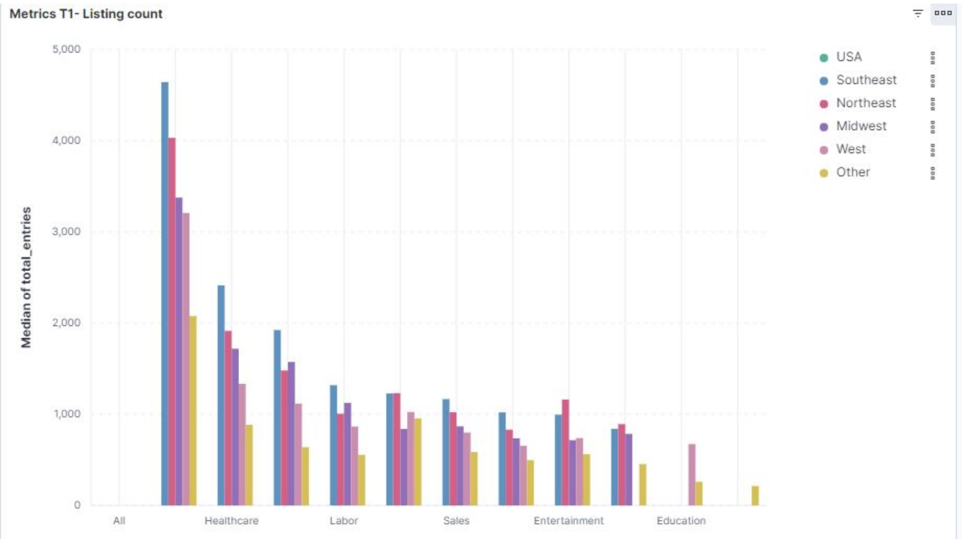
Requirements: calculate the average salary and job listing counts per region and category.

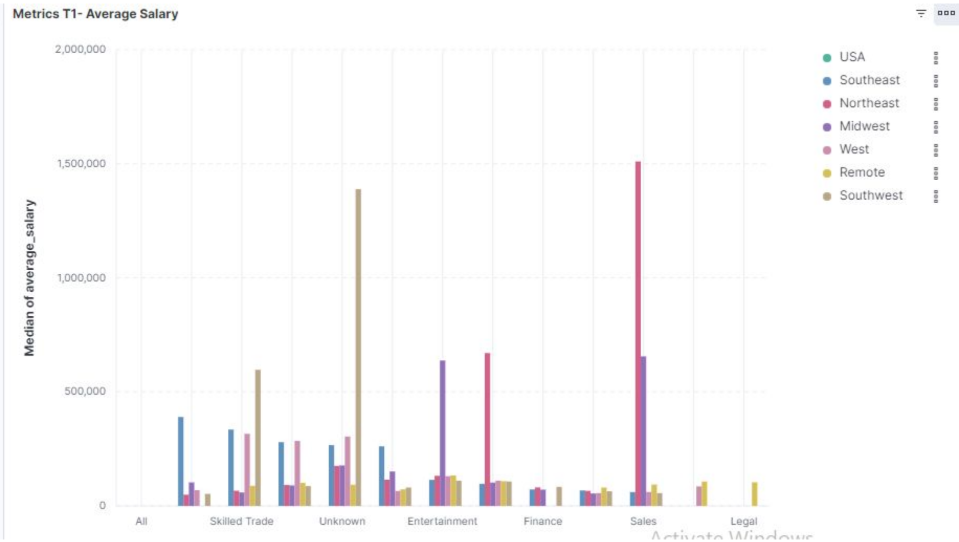
- Average salary per region per category per month
- Listing counts per region per category per month

Metrics generation and evaluation was done on a new index called jobs_health, where the large dataset in jobs_prd was shortened into a smaller set of datapoints representing aggregations of data occurring in each month, category, and region. These fields were generated through aggregations of jobs_prd:

- Category
- Month
- Region
- Average_salary*
- overall_average_salary*
- Total_entries
- timestamp*

Metrics T1- Results





Multilevel Metrics Generation- Tier 2

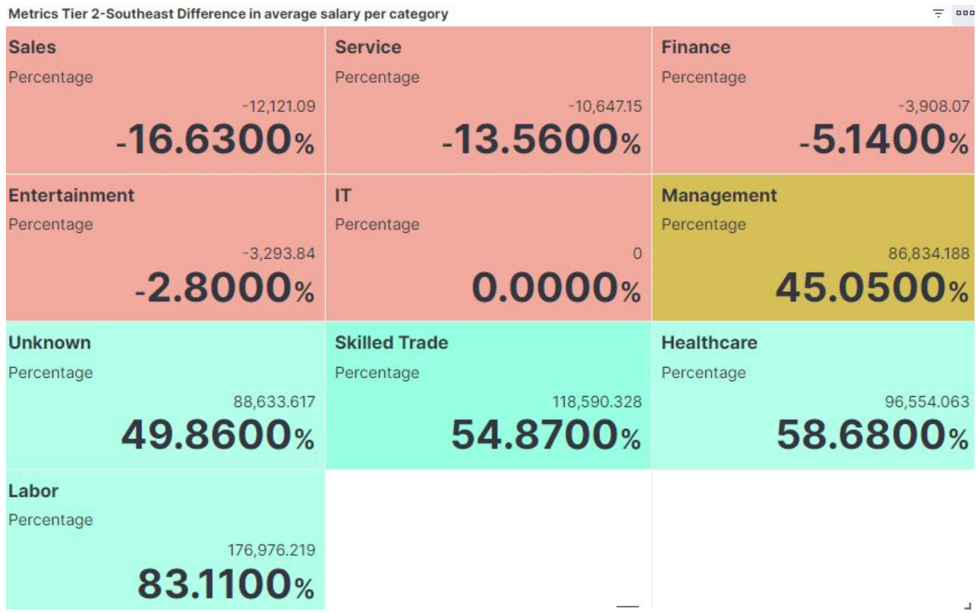
Objective: Based on level-1 metrics, generate comparison metrics, which function better as health indicators. These metrics indicate average salary increases/decreases (number and %) for each job category per region. It can also serve as a method to calculate the SLO (Service Level Objective) result.

Requirements: Based on common health metrics, calculate the following differential metrics:

1. Difference:
   - Determine the difference between the average salary for each month and the overall average salary.
2. Percent Change:
   - Calculate the percent change between the average salary for each month and the overall average salary.
3. Score:
   - Score will be 0 if the percent change is 0
   - Score will be -1 if the percent change is negative
   - Score will be 1 if the percent change is positive
4. Status:
   - Up if score is 1
   - Down if score is -1
   - Unchanged if score is 0

These new metrics were generated for all documents in jobs_health in fields called “diff” and “diff_percentage”

Metrics T2- Results



Metrics Tier 2-Southeast Difference in average salary per category

Sales
Percentage
-12,121.09
-16.6300%

Service
Percentage
-10,647.15
-13.5600%

Finance
Percentage
-3,908.07
-5.1400%

Entertainment
Percentage
-3,293.84
-2.8000%

IT
Percentage
0
0.0000%

Management
Percentage
86,834.188
45.0500%

Unknown
Percentage
88,633.617
49.8600%

Skilled Trade
Percentage
118,590.328
54.8700%

Healthcare
Percentage
96,554.063
58.6800%

Labor
Percentage
176,976.219
83.1100%

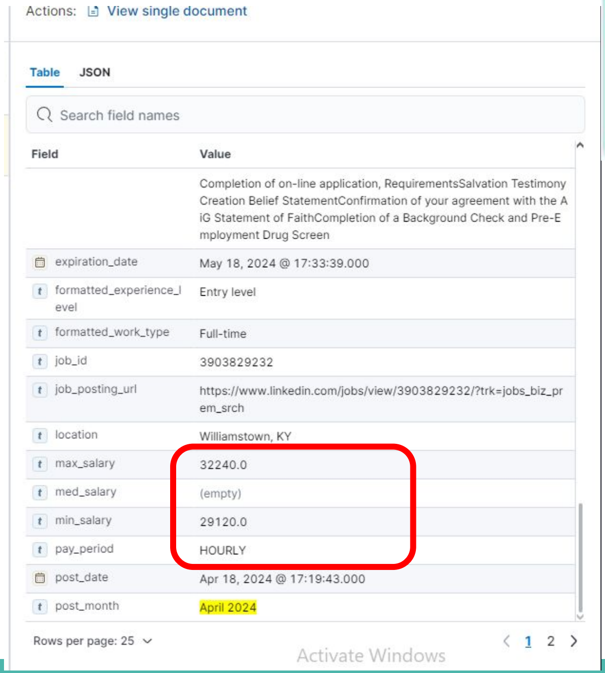
Issue Encountered	Solution
When using a loop to update documents and find the difference between a field and aggregation result, sometimes the correct data can't be inserted into the index, despite the correct result being printed after the initial calculation.	Initialized and indexed a field called overall_average_salary, where the and used a simpler second loop to make the calculation.

Time to process:

- Difference: 0.043248891830444336 seconds
- Percentage Difference: 0.045384168624877930 seconds
- Score and status : 0.043051719665527344 seconds

Anomalies

The best way to prevent anomalies is to take extra measures when cleaning data during the parsing phase. In this case, a more improved script would remove outlying data such as this one.



Actions: View single document

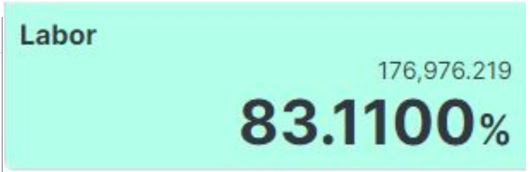
Table JSON

Search field names

Field	Value
	Completion of on-line application, RequirementsSalvation Testimony Creation Belief StatementConfirmation of your agreement with the A iG Statement of FaithCompletion of a Background Check and Pre-E mployment Drug Screen
expiration_date	May 18, 2024 @ 17:33:39.000
formatted_experience_level	Entry level
formatted_work_type	Full-time
job_id	3903829232
job_posting_url	https://www.linkedin.com/jobs/view/3903829232/?trk=jobs_biz_prem_srch
location	Williamstown, KY
max_salary	32240.0
med_salary	(empty)
min_salary	29120.0
pay_period	HOURLY
post_date	Apr 18, 2024 @ 17:19:43.000
post_month	April 2024

Rows per page: 25

1 2



Labor

176,976.219

83.1100%

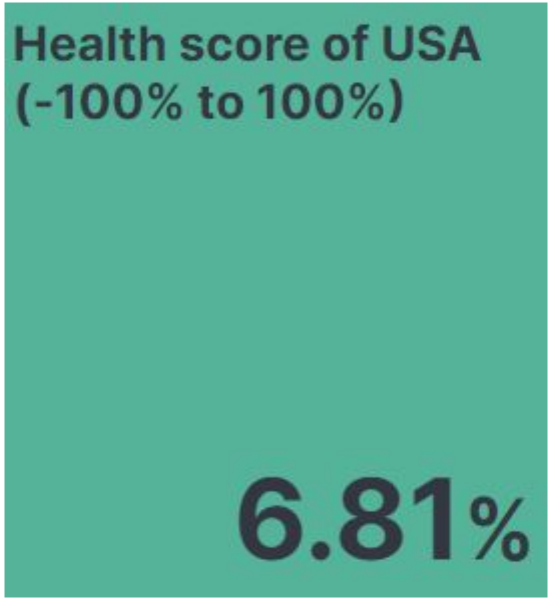
Multilevel Metrics Generation- Tier 3

Objective: Based on level-2 metrics, calculate an overall health indicator. This represents the overall job health in the proof of concept (POC). It can be used to generate an overall service health indicator, based on the FINOPS proportion.

Requirements: Calculate the overall health metrics (level 3) based on weight and regional percent change:

- For each month and each region, do a sum_diff = sum of the diff.
- If the sum_diff > 0 , set score=1. If the sum<0 set score =-1.
- Calculate the overall_score = score * region_weight / sum (region_weight) . The result shall be a float between 1 to -1.
- Based on the data, the (estimated) weights are:
  - Southeast_w = 1.1
  - Northeast_w = 1.0
  - Midwest_w = 0.8
  - West_w = 0.7
  - Southwest_w = 0.6
  - Remote_w = 0.5
- Total weight = 4.7
- Weights can be user-defined
- (percent change * Weight) / (total weight)
- Current health data = 100% + weighted percentage. If the result is over 100%, the health is 100%, or good condition.

Metrics T3- Results



Health score of USA (-100% to 100%)

6.81%

Conclusion

Business Result
In the month of April 2024, the US job market showed a slight increase, which may be attributed to the anomalies we discussed, the sparse dataset, or the general trend seen in the average salary indicators.
Technical Result

1. **Data Quality**: Data quality is a critical factor. A significant amount of time was spent on parsing and preparing the data. However, there are still numerous opportunities for improvement in this process.
2. **Simplicity in Logic**: Keeping the logic simple is essential, as it simplifies both implementation and troubleshooting, ensuring smoother workflows.
3. **Performance Optimization**: Whenever possible, use native aggregation functions due to their faster computing speeds. If you cannot use native aggregation functions, creating smaller indices can help speed up the process.

Future Improvements

1. Enhance the category based on job title. (when present, you can voice over that because of the different job titles, the category parsing is not idea currently. There are still a lot of documents are not parsing to the correct category.
2. Extra measures when cleaning data, such as removing outliers, which may be the resu
3. Current data is limited and outdated. Use the worm software to grep more up-to-date data.
4. Improve level 3 metrics based on more health metrics, such as listing number changes.

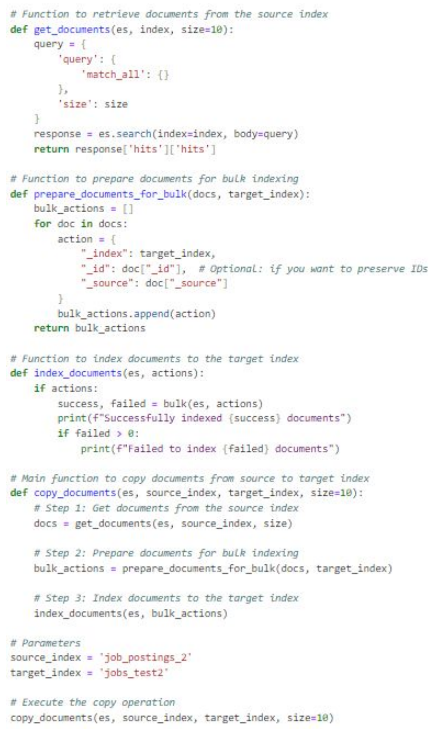
References

https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-bool-query.html

https://www.kaggle.com/datasets/arshkon/linkedin-job-postings/data

Supporting slides

Creating cardinal index



```
# Function to retrieve documents from the source index
def get_documents(es, index, size=10):
    query = {
        'query': {
            'match_all': {}
        },
        'size': size
    }
    response = es.search(index=index, body=query)
    return response['hits']['hits']

# Function to prepare documents for bulk indexing
def prepare_documents_for_bulk(docs, target_index):
    bulk_actions = []
    for doc in docs:
        action = {
            "_index": target_index,
            "_id": doc["_id"],  # Optional: if you want to preserve IDs
            "_source": doc["_source"]
        }
        bulk_actions.append(action)
    return bulk_actions

# Function to index documents to the target index
def index_documents(es, actions):
    if actions:
        success, failed = bulk(es, actions)
        print(f"Successfully indexed {success} documents")
        if failed > 0:
            print(f"Failed to index {failed} documents")

# Main function to copy documents from source to target index
def copy_documents(es, source_index, target_index, size=10):
    # Step 1: Get documents from the source index
    docs = get_documents(es, source_index, size)

    # Step 2: Prepare documents for bulk indexing
    bulk_actions = prepare_documents_for_bulk(docs, target_index)

    # Step 3: Index documents to the target index
    index_documents(es, bulk_actions)

# Parameters
source_index = 'job_postings_2'
target_index = 'jobs_test2'

# Execute the copy operation
copy_documents(es, source_index, target_index, size=10)
```