DAT535 Project Report

Job Recommendation System

Dominykas Petniunas & Binyam Kebrom Tedella
University of Stavanger, Norway
d.petniunas@stud.uis.no & bk.tedella@stud.uis.no

ABSTRACT

This project develops a Spark-based data pipeline to analyze job descriptions, extract key trends, and enable real-time analytics. The pipeline includes data ingestion, cleaning with MapReduce routines, and serving insights through clustering and keyword extraction algorithms. The outcomes aim to support recruiters, HR professionals, and job seekers by identifying industry trends and aligning job requirements with market expectations.

KEYWORDS

Job Recommendation System, Medallion Architecture, Apache Spark, Data Ingestion, Data Cleaning, Data Serving, MapReduce, MLlib Clustering, TF-IDF, Data Pipeline

1 INTRODUCTION

The increasing volume of job postings and applications necessitates efficient job recommendation systems. This project focuses on creating a scalable and efficient pipeline using Apache Spark to process and analyze job descriptions, ultimately providing relevant job recommendations. Additionally, it aims to analyze unstructured job description data to extract insights into job market trends. By leveraging Apache Spark and MLlib, we aim to identify skill demands, industry trends, and role similarities.

Recommendation system is a tool designed to provide personalized suggestions to users based on their preferences, behavior, and interaction with a platform. This type of recommendation can be found everywhere from gaming and social media to banking and healthcare [4]. In the context of job recruitment, these systems assist seekers in discovering opportunities that matches some specified skills and preferences. The traditional recommendation system are primarily divided into two approaches: collaborative filtering, as utilized by platforms such as Netflix and Youtube and content-based with Amazon utilizing it. [1] [18]

Previous studies have explored several methods for job recommendation systems. For instance, there are several papers describing development of hybrid models combining collaborative filtering and content-based filtering to improve recommendation accuracy [20]. Our approach differs by focusing on scalability and efficiency through the use of Apache Spark. Additionally, our project contributes to the recommendation system field by developing a pipeline that not only provides personalized job recommendations but also offers great valuable into the job market through a custom job recommendation dashboard made in `Power BI`.

Supervised by Tomasz Wiktorski and Jayachander Surbiryala .

Project in Data-intensive Systems and Algorithms (DAT535), 2024.

2 BACKGROUND

Medallion Architecture is a data processing architecture that organizes data into three layers: Bronze, Silver, and Gold. The Bronze layer stores raw, unprocessed data, serving as the initial phase in the implementation. The Silver layer contains cleaned and transformed data, ensuring quality and consistency for algorithm implementation. Finally, the Gold layer keeps the refined data, optimized for analysis. Apache Spark is our main tool for implementation which is a powerful open-source processing engine for big data, known for its speed and ease of use.

3 METHOD

Continuing from the Background section, Apache Spark is a distributed computing framework suitable for large-scale data processing. It enables efficient handling of big data through its Resilient Distributed Datasets (RDDs) and DataFrame abstractions, alongside machine learning libraries in MLlib.

Clustering and TF-IDF Clustering algorithms like K-means group data points based on similarity facilitates the identification of patterns within unstructured data. TF-IDF quantifies the importance of terms in a document relative to a corpus, enhancing feature extraction for text data. These techniques enable structured insights from unstructured text, but their effectiveness depends on careful parameter tuning and preprocessing.

3.1 Design

The pipeline consists of three stages: data ingestion to load cleaned job data into Spark Data Frames, data preprocessing to filter rows, clean values, and retain relevant columns, and data serving with TF-IDF feature extraction, K-means clustering optimization, and evaluation using the Silhouette Score.

Figure 1 shows a simple design model. As mentioned we used Apache Spark to build a scalable pipeline that processes data through multiple layers. We began with a structured dataset found in Kaggle, which we unstructured to align with the requirements of our project.[16] The data was initially placed in the bronze layer, where the unstructured, raw, and unrefined data is stored. From there we moved to the silver layer, where we performed data preprocessing tasks, including cleaning, standardization, and transformation. After that, we went to the final, gold layer, where we implemented the various recommendation algorithms ad mentioned using Apache Spark's MLlib. Finally, we used Power BI, a visualization tool, to create an interactive dashboard to visualize and populate data, making it accessible for analysis and decision-making.

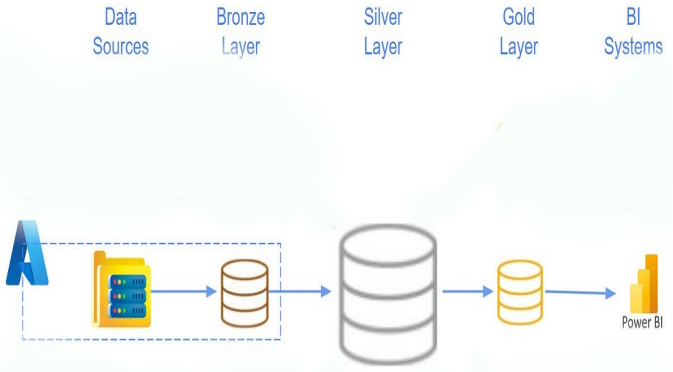


Figure 1: Design Representation

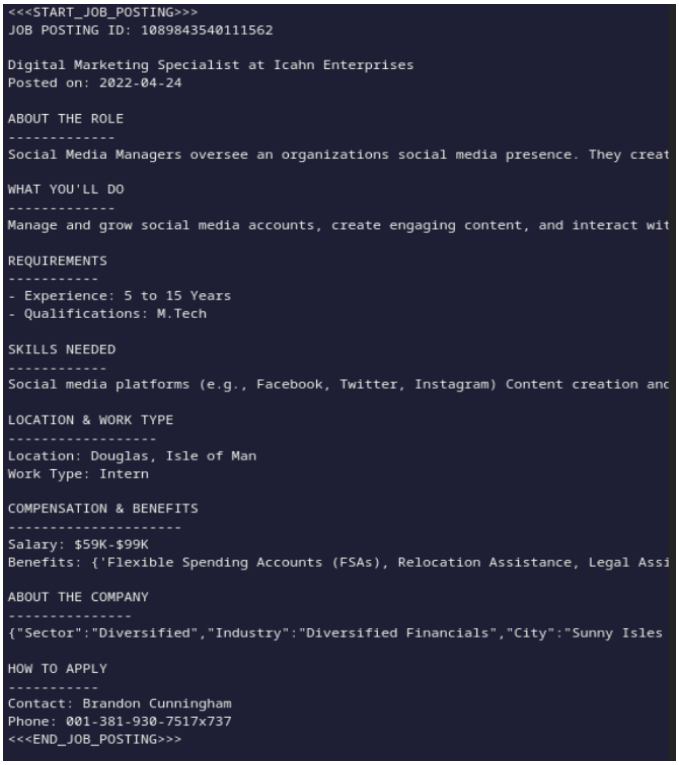
3.2 Implementation

3.2.1 Unstructuring. The unstructuring of the dataset was our initial step in the project, where we transformed the structured data into a raw, unstructured text. This process involved converting tabular comma-separated values file into free-form text. The unstructured data is visualized in the image below 2 as a raw text file, showing how data appears after this transformation.

Listing 1: Spark mapPartition operation implementation [7]

```
# Step 1: MAP - Group lines into job
    postings
def map_to_job_postings(lines):
    current_posting = []
    for line in lines:
        if "<<<START_JOB_POSTING>>>" in line
            :
            if current_posting:
                yield "\n".join(
                    current_posting)
            current_posting = []
        elif "<<<END_JOB_POSTING>>>" in line
            :
            if current_posting:
                yield "\n".join(
                    current_posting)
            current_posting = []
        else:
            current_posting.append(line)

job_postings = raw_data.mapPartitions(
    map_to_job_postings)
```



```
<<<START_JOB_POSTING>>>
JOB POSTING ID: 1089843540111562

Digital Marketing Specialist at Icahn Enterprises
Posted on: 2022-04-24

ABOUT THE ROLE
--------------
Social Media Managers oversee an organizations social media presence. They creat

WHAT YOU'LL DO
--------------
Manage and grow social media accounts, create engaging content, and interact wit

REQUIREMENTS
------------
- Experience: 5 to 15 Years
- Qualifications: M.Tech

SKILLS NEEDED
-------------
Social media platforms (e.g., Facebook, Twitter, Instagram) Content creation anc

LOCATION & WORK TYPE
--------------------
Location: Douglas, Isle of Man
Work Type: Intern

COMPENSATION & BENEFITS
-----------------------
Salary: $59K-$99K
Benefits: {'Flexible Spending Accounts (FSAs), Relocation Assistance, Legal Assi

ABOUT THE COMPANY
-----------------
{"Sector":"Diversified","Industry":"Diversified Financials","City":"Sunny Isles

HOW TO APPLY
------------
Contact: Brandon Cunningham
Phone: 001-381-930-7517x737
<<<END_JOB_POSTING>>>
```

Figure 2: Unstructured free-format raw text.

The raw text in the image [2] represents a job posting formatted with key details necessary for the Gold layer such as job title, company name, job description and more. It is structured in a sense that it is divided into some section such "ABOUT THE ROLE", "REQUIREMENTS", and "SKILLS NEEDED", but lacks any clear organization or structure typical of more standardized data formats.

Since unstructuring the data was not part of the evaluation process, it was mainly accomplished with the assistance of ChatGPT. Additionally, AI tools, such as ChatGPT, were used throughout the development process to troubleshoot and get guidance when functions did not behave as expected, often providing long outputs that were "hard to analyze". [14]

3.2.2 Data Cleaning. In this sub-section, we describe the data cleaning and preprocessing steps implemented using basic MapReduce routines in Spark. This process was crucial for transforming raw, unstructured text into a clean dataset ready for algorithm implementation. The two listings show the proper implementation used for the main Spark operations in the silver layer.[5, 8]

Listing 2: Spark mapPartition operation implementation [5, 8]

```
# Step 1: MAP - Group lines into job
    postings
def map_to_job_postings(lines):
    current_posting = []
    for line in lines:
        if "<<<START_JOB_POSTING>>>" in line
            :
            if current_posting:
                yield "\n".join(
                    current_posting)
            current_posting = []
        elif "<<<END_JOB_POSTING>>>" in line
            :
            if current_posting:
                yield "\n".join(
                    current_posting)
            current_posting = []
        else:
            current_posting.append(line)

job_postings = raw_data.mapPartitions(
    map_to_job_postings)
```

Listing 3: Main Spark operations from the silver layer implementation. [15]

```
    # Extract and clean fields
    cleaned_postings = job_postings.map(
        extract_fields).filter(lambda x: x
        is not None)


    # Remove duplicates based on all fields
    unique_postings = cleaned_postings.
        distinct()


    # Calculate statistics
    total_posts = unique_postings.count()
    posts_with_missing = cleaned_postings.
        filter(lambda x: None in x).count()
```

3.2.3 Data cleaning challenges. This approach allowed us to handle large datasets efficiently, leveraging Spark's fast computational capabilities. [10] During the data-cleaning implementation we came across several challenges and a lot of time were spent trying to properly solve them. The first challenge was transitioning from familiar libraries like Pandas and NumPy to Spark-only libraries.

Another critical challenge was duplicate removal. Initially, we implemented a simple duplicate check that looked for identical records. Without overthinking we assumed it worked as intended. However, as we continued implementing the algorithm we obtained results that did not make sense. Upon closer examination, we discovered that the records have identical values except for some missing letters, such as "yrs" instead of "years" and "min" instead of "minimum", among others. We addressed this by standardizing terms, which helped to properly reduce duplicates effectively.

Lastly, our unstructuring process resulted in over 70 million lines of text, leading to slow processing and memory constraints. To mitigate this, we divided the data into batches of 1000 records each [4], optimizing memory usage and improving processing speed by enabling parallel processing [11]. This batching approach was crucial in managing the large volume of data efficiently since we had to create several cleaned datasets with different columns for algorithm testing. The listings below shows the main implementation of how batching was performed and the minor output showing total records and batch initialization.

Listing 4: Batch Processing Function [14]

```
def process_csv(input_file, output_file,
  batch_size=1000):

  chunks = pd.read_csv(input_file,
  chunksize=batch_size)

  for chunk in tqdm(chunks,
  desc="Processing"):

      processed_batch =
      process_batch(chunk)

      processed_dfs.append(processed_batch)

  final_df = pd.concat(processed_dfs,
  ignore_index=True)

  final_df.to_csv(output_file, index=False)
```

Listing 5: Batching minimized output

```
Starting CSV export at 08:39:38
Total records to process: 1,610,355
Processing in 323 batches of 5000 records
    each

Processing batch 1/323 (0 to 5,000 records)

Batch 1 complete - 5,000 records written

Processing batch 2/323 (5,000 to 10,000
    records)
```

3.2.4 The Golder Layer. The process began with reading the raw text file and grouping lines into individual job posting using the `mapPartitions` spark function. We then applied `map` function to stardardize the terms such as "yrs" to "years", "min" to "minimum", and more to ensure consistency. Invalid entries were removed using the `filter` function, and duplicates were eliminated with the `distinct` function.

The data is loaded into Spark as follows:

Listing 6: Loaded Data.

```
# Initialize Spark Session
spark = SparkSession.builder \
    .appName("SkillsDemandAnalysis") \
    .config("spark.executor.memory", "4g") \
    .config("spark.driver.memory", "4g") \
    .getOrCreate()

# Load cleaned data as DataFrame
data_path = "hdfs://namenode:9000/user/
    ubuntu/Project_Code/
    cleaned_job_data_using_spark.csv"

job_data = spark.read.csv(data_path,
header=True, inferSchema=True)

# Select required columns and
# preprocess missing data
selected_columns = ["job_description",
                    "job_skills",
                    "job_industry",
                    "job_location",
                    "experience_level",
                    "min_salary",
                    "max_salary"]
```

Feature Extraction and Initial Clustering We applied TF-IDF and K-means clustering and The initial Silhouette Score was -0.034, indicating poor clustering performance.

Listing 7: Silhouette Score Initial.

```
# Evaluate clustering using silhouette score
# The elbow method calculates the within-
# cluster sum of squares (WCSS) for various
# values of k and identifies the "elbow"
# point.

# Rename 'cluster' column to 'prediction'

# Use 'cluster' column as the
# prediction column

clustered_data = clustered_data.
    withColumnRenamed(
    "cluster", "prediction")

# Evaluate clustering using silhouette score
evaluator = ClusteringEvaluator(
featuresCol="features",
predictionCol="prediction",
metricName="silhouette")

silhouette_score = evaluator.evaluate(
    clustered_data)

[Stage 42:================================>
(5 + 3) / 8]

Silhouette Score: -0.034
```

3.3 Optimization and Improved Implementation

We made several improvements to address the low Silhouette Score. We added Stop Words Remover to eliminate common words that do not contribute to meaningful clusters. We calculated the Silhouette Score to assess the quality of the clusters. After optimization, the silhouette score increased to 0.589, indicating a substantial improvement in cluster cohesion and separation.

Listing 8: Evaluating Clustering

```
# Apply KMeans clustering
kmeans = KMeans(k=5, seed=42,
featuresCol="features", predictionCol="
    prediction")

kmeans_model = kmeans.fit(clustered_data)

clustered_data = kmeans_model.transform(
    clustered_data)

# Evaluate clustering using silhouette score
```

```
evaluator = ClusteringEvaluator(featuresCol=
    "features",
predictionCol="prediction", metricName="
    silhouette")

silhouette_score = evaluator.evaluate(
    clustered_data)

print(f"Silhouette Score: {silhouette_score
    :.3f}")

Silhouette Score : 0.589
```

4 EXPERIMENTAL EVALUATION

4.1 Experimental setup

The experimental setup for our pipeline inolved deploying an Ubuntu virtual machine within an OpenStack environment. This VM served as the master node, running a single `NameNode`, and three worker nodes `DataNodes` which collectively formed a Hadoop Distributed File System (HDFS) [2]. The experiments ran on Spark 3.0, configured to leverage the resources across the nodes having 8 GB of RAM allocated.

To evaluate the pipeline's scalability, we also conducted tests using Amazon Web Services (AWS). Specifically, we transferred several datasets from our AWS S3 bucket to our Spark environment. This was achieved using the AWS CLI with the following command

```
aws s3 cp s3://project-spark-bucket1/
    job_descriptions.csv .
```

Additionally, several errors were encountered during the initial testing phase, which led us to experimentation with different Spark configurations. We used various Spark configurations depending depending on the specific implementation we were working on. For small-scale jobs, we used `master("local[*]")`, as it runs on a single machine and has less overhead compared to managing a cluster [9]. For large-scale jobs, we utilized configurations like `master("yarn")` or `.set("spark.default.parallelism", "100")` to take advantage of distributed computing.

Listing 9: Local Spark Configuration

```
from pyspark.sql import SparkSession

# Restart the Spark session
spark = SparkSession.builder \
    .appName("SkillsDemandAnalysis") \
    .master("local[*]") \
    .config("spark.executor.memory", "4g") \
    .config("spark.driver.memory", "4g") \
    .getOrCreate()
```

4.2 Results

Our project leverages the Medallion Architecture to process job market, focusing on transforming the Silver Layer into the Golden Layer to prepare data for skilled job analysis. Key results include clustering job descriptions, job_skills, job_industry, job_location, experience_level, min_salary and max_salary. These clusters serve as the foundation for analyzing trends and visualizing insights for the skilled job market. The final output is structured and ready for visualization in a dashboard, providing actionable insights for job seekers, HR professionals,and policymakers[3, 13]

4.2.1 ***Key Results.*** • Clustering Performance: Initial clustering yielded a Silhouette Score of -0.034, indicating poorly defined clusters with overlapping and misclassified data points. Challenges included low numFeatures in HashingTF, ineffective handling of common words, and insufficient text standardization. After optimization, the Silhouette Score improved to 0.589, reflecting well-defined clusters with good separation. Improvements included increasing numFeatures to better capture unique terms, removing stop words to reduce noise, and consolidating preprocessing into a single pipeline for consistency and efficiency.[9, 19]

• Clustering Profile: Clusters were profiled by analyzing frequent terms in job descriptions. Tokenized words were exploded into rows, grouped by clusters, and analyzed for word frequency. This highlighted distinctive skills and trends in each cluster, aiding skill matching and trend analysis. [6, 19]

• Clustering Optimization: The optimal number of clusters was determined using the elbow method[12] and silhouette scores, balancing interpretability and cluster quality. The visualization of silhouette scores showed clear improvement after optimization.

• Automated Pipeline: A spark pipeline automated tokenization, feature extraction, and clustering, enabling scalable and repeatable workflows for processing large data sets efficiently.[9, 17]

4.2.2 ***Significance.*** • Job Market Analysis: Clustering reveals demand patterns by grouping similar job descriptions, helping stakeholders understand trends

• Skill Matching: Identifies skills associated with job types, improving recommendations for job seekers.

• Scalability: Distributed processing with Apache ensures the workflow is efficient for large-scale data

4.3 Discussion

This project demonstrated the effectiveness of integrating Apache Spark with The Medallion Architecture to analyze large-scale job market data. The clustering approach provided a structured understanding of job descriptions, revealing key trends and demands in the market. By focusing on tokenization and feature extraction, the project addressed the challenges of processing unstructured text data efficiently. The use of evaluation metrics underscored the value

of robust preprocessing and parameter tuning.[9] The clustering results helped group job descriptions into categories that showcased distinct characteristics, such as industry-specific terminology and geographic trends. For example, frequent terms in pharmaceutical clusters highlighted skills like clinical research, while clusters for manufacturing building systems pointed to design and technical expertise. Challenges in feature representation and interoperability of clusters were mitigated through iterative profiling and refinement, making the results reliable and insightful.[6, 19] Despite its strength, the project faced challenges in identifying the optimal number of clusters and ensuring the interpretability of practical use. However, the integration of profiling techniques, such as identifying top terms in the cluster, bridges this gap, making the analysis more actionable for stakeholders.

4.4 Dashboard Analysis

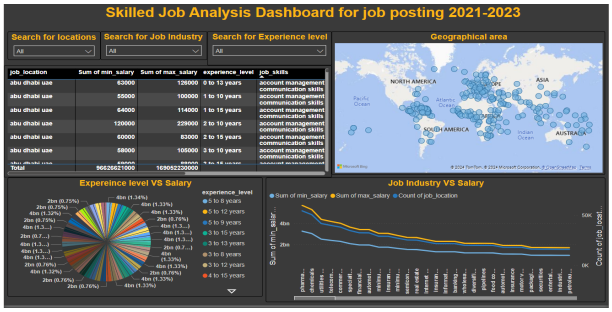


Figure 3: Dashboard interaction.

This analysis evaluates job opportunities across industries and geographical locations, leveraging data from a clustering model to uncover trends in job availability and skills demand. The primary use case is to provide insights for recommending jobs to users based on descriptions, locations, and salary ranges. These job descriptions data sets shows the job posted during sept 2021-2023 Even though we don't need to create the dashboard we would like to give insight into our analysis. We used our ready data for visualization using Power BI.

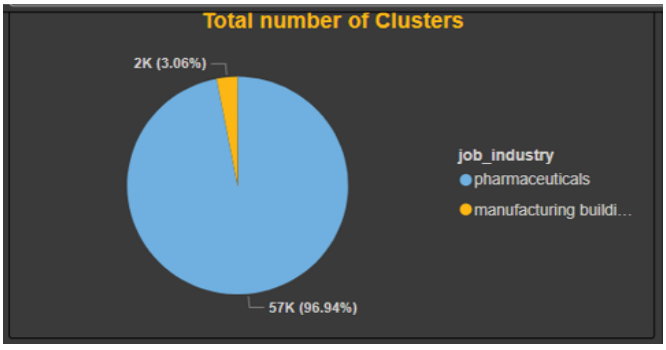


Figure 4: Job by cluster.

The pharmaceutical industry leads with 57,000 jobs, showing diverse demand globally, with Harare, Zimbabwe, having the lowest

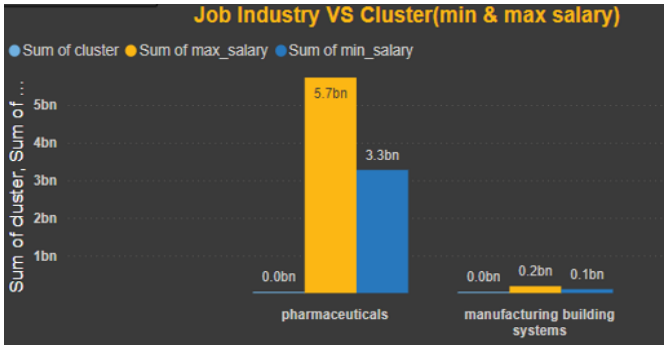


Figure 5: Job by cluster min and max salary.

count (217). Manufacturing building systems is the smallest cluster with 1,790 jobs, and San Juan, Puerto Rico, records the lowest count (2). The high job activity is concentrated in North America, Europe, and Asia, while parts of Africa and island nations show minimal activity. Pharmaceuticals offer wide salary ranges across experience levels while manufacturing building systems show limited ranges. Recommendations include targeting North America and Europe for pharmaceutical jobs and focusing on niche roles for manufacturing building systems. Enhancements like personalized job recommendations, salary insights, and regional filters are advised. Future analysis should explore skill-level trends, emerging industries, and real-time data monitoring.

5 FUTURE WORK

For future work, it would be great to implement and compare deep `learning` and `reinforcement learning` approaches to enhance the accuracy of job recommendations.

Deep learning, as a form of content-based filtering, has the potential to improve recommendation systems by analyzing unstructured data such as job descriptions, resumes, and user interactions. Techniques such as natural language processing (NLP) could be employed to better understand the content of job postings and candidate profiles, allowing for more accurate recommendations. For example, deep learning models could learn to identify patterns in job titles, descriptions, and skill requirements, thereby improving job matching.

Reinforcement learning offers delivery of recommendations by experimenting and learning in a feedback loop. In this framework, the recommendation system would experiment with different job suggestions and learn from user responses. For example, if a user engages positively with a specific job posting, the system could prioritize similar jobs in future recommendations. Over time, reinforcement learning could optimize the recommendation process, tailoring job suggestions based on user preferences and behaviors.

Even though these are some major concepts, these methods would provide valuable insights into how advanced machine learning

techniques can enhance the job recommendation system, making it more dynamic and responsive to evolving user needs.

6 CONCLUSION

This report demonstrates the successful development of a scalable and efficient pipeline for analyzing job market trends, leveraging Apache Spark, NLP techniques, and clustering algorithms. Initial challenges with clustering performance, such as poorly defined clusters and overlapping data points were addressed through enhanced feature extraction and refined data preprocessing. These improvements resulted in significantly better clustering outcomes, providing well-defined groups and actionable insights.

The analysis highlights the pharmaceutical industry's dominance in job count and identifies manufacturing building systems as a niche market with limited opportunities. Geographic disparities were also uncovered, revealing regions with significantly fewer job postings, and offering insights into localized job market dynamics. The clustered data was prepared for dashboard visualization, enabling stakeholders to explore high-demand skills, industry-specific trends, and salary distributions interactively.

Furthermore, the results provide a robust foundation for building an optimized recommendation system. By aligning user profiles with market trends, this system can effectively match candidates with opportunities that suit their skills, preferences, and career goals. This project not only showcases the power of big data technologies but also demonstrates their potential to transform the way job market data is analyzed and utilized for strategic decision-making. Future enhancements, including real-time data integration and advanced NLP techniques, will further refine the analysis and expand its applicability in dynamic and evolving job markets.

REFERENCES

[1] Abhinav Ajistaria. 2019. Build a Recommendation Engine With COllaborative Filtering. https://realpython.com/build-recommendation-engine-collaborative-filtering/?utm_source=chatgpt.com

[2] Dhruba Borthakur. May 18, 2022. HDFS Architecture Guide. https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

[3] Databricks. 2023. Medallion Architecture. Available at: https://www.databricks.com/glossary/medallion-architecture.

[4] Cem Dilmegani. Nov 19, 2024. Recommendation Systems: Applications and Examples. https://research.aimultiple.com/recommendation-system/

[5] Apache Spark Documentation. 2023. Batch Processing in Spark. Available at: https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html.

[6] Apache Spark Documentation. 2023. Cluster Profiling with Spark. Available at: https://spark.apache.org/docs/latest/ml-clustering.html.

[7] Apache Spark Documentation. 2023. MapPartitions Transformation in Spark. Available at: https://spark.apache.org/docs/latest/rdd-programming-guide.html#transformations.

[8] Syeda Marium Faheem. Jun. 14, 2021. Apache Spark: Data cleaning using PySpark for beginners. https://medium.com/bazaar-tech/apache-spark-data-cleaning-using-pyspark-for-beginners-eeeced351ebf.

[9] Software Testing Help. 2023. TF-IDF and Clustering Overview. Available at: https://www.softwaretestinghelp.com/apache-spark-tutorial/.

[10] Mahdi Karabiben. Dec. 19, 2019. Why Apache Spark Is Fast and How to Make It Run Faster. https://towardsdatascience.com/why-apache-spark-is-fast-and-how-to-make-it-run-faster-9d31bf3eae04

[11] Afreen Khalfe. Aug. 4, 2023. Mastering Batch Processing in Python: From Basic Examples to Advanced Techniques. https://talent500.com/blog/batch-processing-handling-large-volumes-of-data-in-scheduled-or-periodic-batches/

[12] Scikit learn Documentation. 2023. Choosing the Right Number of Clusters with the Elbow Method. Available at: https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html.

[13] Microsoft. 2023. Power BI Documentation. Available at: https://learn.microsoft.com/en-us/power-bi/.

[14] OpenAI. 2023. ChatGPT Documentation. Available at: https://openai.com/chatgpt.

[15] Hari Prasad. Jan. 20, 2024. DropDuplicate, Distinct and GroupBy in Apache Spark | Efficiently Remove Duplicates/Redundant Data. https://medium.com/@hprasad98/dropduplicate-distinct-and-groupby-in-apache-spark-7f9a6501d000.

[16] Ravender Singh Rana. [n. d.]. Job Dataset. https://www.kaggle.com/datasets/ravindrasinghrana/job-description-dataset/data

[17] Apache Spark Team. 2023. Apache Spark Overview. Available at: https://spark.apache.org/.

[18] The Upwork Team. [n. d.]. What Is Content-Based Filtering? Benefits and Examples in 2024. https://www.upwork.com/resources/what-is-content-based-filtering?utm_source=chatgpt.com

[19] Analytics Vidhya. 2023. Named Entity Recognition Techniques. Available at: https://www.analyticsvidhya.com/blog/2021/05/understanding-named-entity-recognition-ner-and-it-applications/.

[20] Raden Ibnu Huygenz Widodo. July 22, 2024. Job Recommendation System Combining Collaborative Filtering and Content Based Filtering. https://www.preprints.org/manuscript/202407.1700/v1