**Conestoga College**

School of Applied Computer Science & Information

Technology

SENG8081 - Case Studies

**Canadian Job Market Trends Analysis**

Anuroopa Balachandran

Bin Hu

Ce Chen

Xiaoman Yang

June 22, 2025

**Abstract**

In today’s rapidly evolving economic landscape, data-driven insights into job market trends are essential for policymakers, businesses, and job seekers to make informed decisions. This project presents a comprehensive analysis of the Canadian job market by integrating real-time job postings with historical labour force statistics from government APIs and open datasets. Leveraging a robust Big Data pipeline, the system processes and correlates key metrics—including employment rates, sectoral growth, regional disparities, and emerging skill demands—to uncover actionable trends.

By combining structured labour market data (Statistics Canada) with unstructured job posting descriptions (scraped from industry platforms), the project identifies high-growth industries, regional employment hotspots, and shifting skill requirements. Advanced analytics and interactive dashboards (Tableau/Power BI) visualize these trends, enabling stakeholders to detect economic signals, anticipate labor shortages, and assess recovery patterns post-recession.

The solution emphasizes scalability, with automated data collection (Python, APIs), cloud-based storage (SQL Server, Azure), and machine learning-ready preprocessing (NLP for skill extraction). Ultimately, this project serves as a foundation for evidence-based workforce planning, helping bridge the gap between labor supply and demand in Canada’s dynamic economy.

**Introduction**

This project analyzes trends in Canada’s job market by integrating real-time job postings with historical labour force statistics. Our goal is to identify high-growth sectors, regional employment disparities, and emerging skill demands to support policymakers, businesses, and job seekers.

**Big Data System Diagram**

Below is an overview of our data pipeline:

[Government APIs (Statistics Canada)]

↓

[Python Backend: Data Ingestion & Cleaning]

↓

[SQL Server: Structured Storage]

↓

[Tableau/Power BI: Visualization & Dashboarding]

Key components:

* **Data Sources**: Job postings (web-scraped/Kaggle) + labour force stats (government APIs).
* **Processing**: Python (Pandas, SQLAlchemy) for cleaning/merging.
* **Storage**: SQL Server for structured time-series data.
* **Output**: Interactive dashboards for trend analysis.

**Data Research and Integration**

**Data Sources**

* **Job Postings (**CLEANED\_job\_postings.csv**)**:
  + Fields: Job title, company, location, salary, sector, work type, descriptions.
  + Source: Web-scraped/Kaggle (cleaned for duplicates/missing values).
* **Labour Force Stats (**CLEANED\_labour\_force\_stats.csv**)**:
  + Fields: Employment rates, demographics, regional data (2020–2023).
  + Source: Statistics Canada API (seasonally adjusted).

**Integration Plan**

* Merge datasets on **location** (province) and **time** (month/year).
* Use **sector** (e.g., IT, Healthcare) to align job demand with labour supply trends.

**Data Collection**

**Data Sources & Tools**

1. **Job Postings Dataset (**CLEANED\_job\_postings.csv**)**
   * **Source**: Aggregated from multiple platforms:
     + **Kaggle**: Pre-cleaned job postings dataset for Canadian market.
     + **Web Scraping**: Python (BeautifulSoup, Scrapy) to extract real-time postings from LinkedIn, Indeed, and company career pages.
   * **Fields Collected**:
     + Job title, company, location, salary range, sector (e.g., IT, Healthcare), work type (remote/hybrid), and detailed descriptions.
     + Metadata: Posted date, applications count, experience level.
2. **Labour Force Statistics (**CLEANED\_labour\_force\_stats.csv**)**
   * **Source**: Statistics Canada’s Labour Force Survey (LFS) API ([Table 14-10-0287](https://www150.statcan.gc.ca/t1/tbl1/en/tv.action?pid=1410028701)).
   * **Fields Collected**:
     + Monthly employment rates, population estimates (ages 15+), and breakdowns by province, gender, and industry (NAICS codes).
     + Seasonally adjusted metrics for trend consistency.

**Collection Process**

* **Automated Pipelines**:
  + **Job Postings**: Scheduled Python scripts run weekly to scrape and append new postings to the dataset.
  + **Labour Stats**: API calls via requests library to fetch monthly updates; data validated against historical records.
* **Challenges & Solutions**:
  + **Inconsistent Formats**: Job titles/salaries varied (e.g., "$80K–$100K" vs. "competitive").
    - *Solution*: Standardized salary ranges to hourly/yearly rates using regex; imputed missing values with sector medians.
  + **API Limits**: Statistics Canada throttled frequent requests.
    - *Solution*: Implemented caching with SQLite to store interim results and avoid redundant calls.

**Data Storage and Maintenance**

**Storage Architecture**

1. **Database**: Microsoft SQL Server
   * **Schema Design**:
     + **Fact Tables:** job\_postings (fields: job\_id, company\_id, posted\_date, salary), labour\_stats (fields: geo, ref\_date, employment\_rate).
     + **Dimension Tables**: companies, locations, sectors for normalization.
   * **Partitioning**: Tables partitioned by ref\_date (month/year) for faster time-series queries.
2. **Cloud Backup**:
   * Azure Blob Storage for raw CSV backups (retained for 2 years per Canadian data retention policies).

**Maintenance Workflow**

* **Data Cleaning**:
  + **Daily Jobs**: Python scripts flag outliers (e.g., salaries > 99th percentile) for review.
  + **Text Processing**: NLP pipelines (spaCy) extract skills from job descriptions; stored in a skills lookup table.
* **Version Control**:
  + Schema changes logged via Flyway migrations.
  + Monthly snapshots archived to Azure for audit trails.

**Scalability Considerations**

* **Current Storage**: ~10GB (job postings + labour stats for 2020–2023).
* **Future Scaling**:
  + **Horizontal Partitioning**: Split job\_postings by province if data grows beyond 50GB.
  + **Cost Optimization**: Move cold data (pre-2022) to Azure Data Lake for cheaper long-term storage.

**Unit Testing**

**Testing Strategy**

To ensure the reliability of our data pipeline and analysis components, we implemented unit tests for:

1. **Data Validation**
   * Verify structure of incoming job postings (required fields: title, company, location)
   * Validate Statistics Canada API response formats
2. **Transformation Logic**
   * Test salary range normalization (e.g., "$80K/year" → 80000)
   * Check geographic data standardization ("Montréal, QC" → "Quebec")
3. **Analysis Functions**
   * Validate trend-calculation algorithms (e.g., monthly growth rates)
   * Test NLP skill-extraction from job descriptions

**Tools & Implementation**

* **Framework**: pytest (Python)
* **Test Cases**:

*def test\_salary\_normalization():*

*assert normalize\_salary("$90,000/year") == 90000*

*assert normalize\_salary("60K") == 60000*

*assert normalize\_salary("unknown") == None*

*def test\_location\_standardization():*

*assert standardize\_location("Toronto, ON") == "Ontario"*

*assert standardize\_location("VANCOUVER") == "British Columbia"*

* **Coverage**: Aimed for ≥80% code coverage (measured via pytest-cov)

**Challenges & Solutions**

| **Challenge** | **Solution** |
| --- | --- |
| Mocking API calls for testing | Used pytest-mock to simulate Statistics Canada responses |
| Testing NLP components | Created a sample job description dataset with labeled skills |
| Binary file (.docx) diffing | Added smoke tests to verify file existence/size instead of content |

**Automation**

* Tests run automatically via GitHub Actions on every git push
* Failure notifications sent to Slack (#data-quality channel)

**Integration with Existing Sections**

1. **Add to Data Quality**: Unit tests provide the first layer of data validation, catching format errors before they propagate through the pipeline.
2. **Add to Project Timeline**:

| **Date** | **Deliverable** |
| --- | --- |
| Jul 11 | Implement core unit tests |
| Jul 18 | Achieve 80% test coverage |

1. **Add to Team Roles**:

**QA Engineer**: Implements pytest cases, maintains test coverage reports.