PySpark

MINI – PROJECT

Statistical Analysis

of

Campus Recruitments

**Table of Contents**

1. **Introduction**  
   1.1. Setup Checklist for Mini Project  
   1.2. Instructions
2. **Problem Statement**  
   2.1. Objective  
   2.2. Abstract of the Project  
   2.3. Technology Used
3. **Data Modeling and Design**  
   3.1. Data Warehouse Design  
   3.2. OLAP Schema  
   3.3. Schema Relationships
4. **Data Visualization**
5. **Setting up the Environment**

5.1 Transitioning from Jupyter lab to Databricks for PySpark Workflows

5.1.1.Setting Up Your Databricks Environment

5.1.2.Modifying PySpark Code for Databricks

5.1.3.Creating Interactive Dashboards in Databricks

5.2 importing Packages

1. **Implementation in PySpark LOT**  
   6.1. Data Ingestion and Transformation  
   6.2. Dimension Table Creation  
   6.3. Fact Table Creation  
   6.4. Data Population Process  
   6.5. Report Generation
2. **Reports to be Built**  
   7.1. Year Wise College Ranking  
   7.2. Year Wise College Wise Percentage Selection  
   7.3. Year Wise Quarter Wise Placement  
   7.4. Total Students Selected per Company  
   7.5. College Ranking Year Wise
3. **Challenges and Solutions**  
   8.1. Challenges Faced  
   8.2. Solutions and Workarounds
4. **Conclusion and Future Work**
5. **Appendix (Source Files and Code)**  
   10.1. Source File Overview  
   10.2. PySpark Code for Data Transformation  
   10.3. Output Files and Sample Reports

**1. Introduction**

**1.1 Setup Checklist for Mini Project**

The Statistical Analysis of Campus Recruitments project requires a well-defined environment and a set of tools to ensure smooth execution. This checklist will help you set up the hardware, software, and libraries needed for the project.

Before starting the development, ensure the following prerequisites are met:

**Hardware Requirements:**

For the project to run efficiently, your computer should meet the following hardware specifications:

* Processor: At least an Intel Pentium 90 MHz or higher processor. For optimal performance, it is recommended to use an Intel Pentium 166 MHz or higher.
* Operating System: The project is primarily designed to run on Windows 10 or higher. However, it is also compatible with other operating systems such as Linux and macOS.
* Memory (RAM): A minimum of 4GB of RAM is required. It is advisable to have 8GB or more for smooth execution, especially if you plan to process large datasets.
* Disk Space: Ensure you have enough disk space to store both the source files and output data space.

The following software tools are essential for the successful implementation of this project:

1. **AWS:**
   * "AWS (Amazon Web Services) offers a scalable, secure cloud platform for deploying, managing, and scaling applications, providing cost-effective solutions across computing, storage, and more. "Why you need it: Anaconda simplifies package management and deployment, particularly for big data projects. It comes bundled with Python and various data science libraries, making it easier to get started without needing separate installations**.**
2. **Python:**
   * What it is: Python is the primary language used to implement the project. It will be used for data transformation, report generation, and interfacing with PySpark.
   * Why you need it: Python is a versatile language for data processing and is highly compatible with PySpark for distributed data analysis.
3. **IDE (Integrated Development Environment):**
   * PyCharm: A powerful Python IDE that helps in writing and debugging your Python code efficiently. It provides advanced features like code completion, debugging, and version control.
   * Jupyter Notebook: An interactive computing environment that allows you to run code in a step-by-step manner, visualize outputs, and create a narrative around your analysis. Jupyter is great for experiments and immediate feedback, making it a popular choice for working with PySpark.
   * Why you need it: These IDEs allow for faster development, testing, and visualization of data. Jupyter Notebooks, in particular, is ideal for demonstrating and presenting intermediate results.
4. **PySpark:**
   * What it is: PySpark is the Python API for Apache Spark, a distributed processing engine used to handle large datasets. PySpark allows you to work with big data in a distributed fashion, enabling scalability and speed.
   * Why you need it: As the project involves processing potentially large datasets from multiple sources (colleges, companies, surveys), PySpark will help you process this data in a scalable and distributed manner. It also allows you to create a data warehouse for handling the vast amounts of data efficiently.
5. **Java:**
   * What it is: Apache Spark, and by extension PySpark, requires Java to function. PySpark acts as a wrapper for the underlying Java-based Spark engine.
   * Why you need it: You need to have Java installed on your system for PySpark to work correctly. Java 8 or higher is recommended**.**
6. **Internet Explorer or another Web Browser:**
   * Why you need it: A web browser is necessary for accessing documentation, resources, and running Jupyter Notebooks or interacting with web-based tools.
7. **Data Bricks:**
   * Databricks is a unified analytics platform that combines data engineering, data science, and machine learning, enabling organizations to accelerate innovation by simplifying the process of building and deploying data-driven applications at scale.

**Software Installation Steps:**

1. **Install Python:**
   * Python 3.x is recommended. You can download it from https://www.python.org/, but it's usually included with Anaconda**.**
2. **Install PySpark:**
   * Once Anaconda is installed, open the Anaconda Prompt (Windows) or terminal (macOS/Linux) and use the following command to install PySpark:
   * This will automatically install PySpark and its dependencies.
3. **Install Java:**
   * You can download Java from the Oracle website. Alternatively, you can install OpenJDK.
   * After installation, make sure Java is properly configured by setting the JAVA\_HOME environment variable.
4. **Verify the Installation:**
   * To verify if everything is set up correctly, open a terminal or command prompt and type the following:
     + For PySpark: pyspark –version

A screenshot of a computer

Description automatically generated

* + - For Python: python –version
      1. Python 3.9.11
    - For Java: java –version
      1. Java open jdk 17.0.11 2024-04-16

**2. Problem Statement**

**2.1 Objective**

The objective of this mini-project is to build an **OLAP-based Statistical Analysis System for Campus Recruitments**, where the system will generate insightful reports about campus recruitment trends for students across various colleges, based on different companies and years.

The system will:

* Load data from provided CSV files
* Transform the data into a star/snowflake schema
* Populate fact and dimension tables
* Generate reports based on different metrics like student selection percentage, company-wise student selection, and college rankings over years.

**2.2 Abstract of the Project**

This project processes recruitment and survey data using PySpark, an open-source distributed data processing engine. It uses an OLAP approach for organizing the data and generating reports that offer insights into the campus recruitment process across multiple colleges and companies. The generated reports will help in analysing trends, understanding selection criteria, and providing a detailed ranking of colleges based on various metrics.

**2.3 Technology Used**

* **Anaconda**: For managing Python packages and environment.
* **Python**: Core programming language used in conjunction with PySpark.
* **PySpark**: Big data framework used to process, transform, and analyze large datasets.
* **Jupyter Notebook / PyCharm**: IDEs used for developing and testing the PySpark code.

**3. Data Modelling and Design**

**3.1 Data Warehouse Design**

In this project, we design a **data warehouse schema** using the **star** or **snowflake** model, with dimension tables and fact tables. The data warehouse will allow us to perform various aggregations, grouping, and ranking operations.

**Dimension Tables:**

1. **College Dimension (college\_dim)**:
   * College ID
   * College Name
   * College Location
2. **Company Dimension (company\_dim)**:
   * Company ID
   * Company Name
   * Head Office Location
   * Country
3. **Survey Dimension (survey\_dim)**:
   * Survey ID
   * Survey Name
4. **Time Dimension (time\_dim)**:
   * Time ID
   * Year
   * Quarter
   * Month
   * Month Name
   * Day of Week
   * Date

**Fact Tables:**

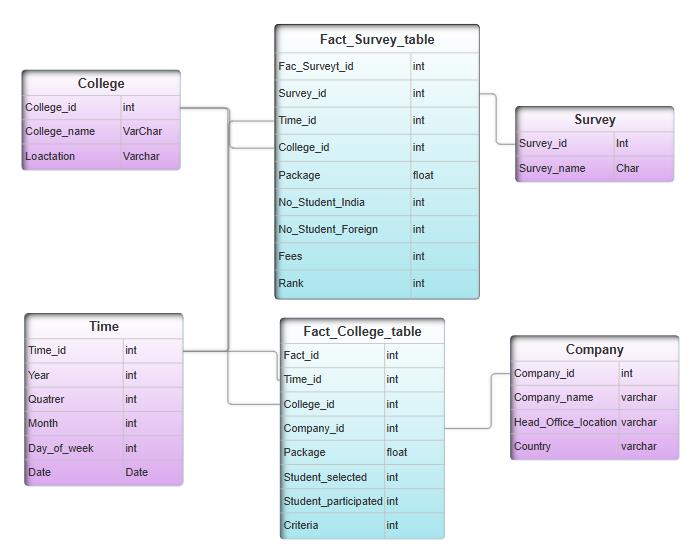
1. **Fact College Details (fact\_college\_details)**:
   * Fact College ID
   * Time ID
   * College ID
   * Company ID
   * Package
   * Students Selected
   * Students Participated
   * Criteria
2. **Fact Survey Details (fact\_survey\_details)**:
   * Fact Survey ID
   * Time ID
   * College ID
   * Survey ID
   * Number of Indian Students
   * Number of Foreign Students
   * Fees
   * Rank

**3.2 OLAP Schema**

The OLAP schema will follow a **star schema model**, with the **fact tables** being at the center, surrounded by the **dimension tables**.

* **Fact Tables** will hold aggregated data, while **Dimension Tables** provide descriptive attributes.
* The relationships between dimension and fact tables are **one-to-many**, where each fact record corresponds to one entry in each dimension table.

**3.3 Schema Relationships**



**4. Data Visualization:**

Databricks Dashboards allow users to create interactive visualizations from SQL queries, notebooks, or data analysis results. Supported visualization types include bar charts, line charts, scatter plots, pie charts, and more. These visualizations can be organized into customizable dashboards, providing a comprehensive view of key metrics. Users can add interactivity through filters, time-range selectors, and cross-filtering between charts. Dashboards can be shared with collaborators, and data can be refreshed on a schedule for real-time insights. Best practices include selecting the appropriate chart type, simplifying designs, and ensuring consistency for clarity. Databricks also integrates with external tools like Tableau and Power BI for enhanced reporting and analytics.

A screenshot of a table

Description automatically generated

A diagram of a pie chart

Description automatically generated

**College Wise – Yearly Ranking:**

**A document with numbers and letters

Description automatically generated with medium confidence**

A graph of a number of people

Description automatically generated with medium confidence

**Quarter Wise – Placement Of Colleges:**

A screenshot of a college list

Description automatically generated

A group of colorful circles with text

Description automatically generated

**5. Setting up the Environment:**

**5.1 Transitioning from Jupyter Lab to Databricks for PySpark**

**Workflows**

When migrating from Jupyter Lab to Databricks, several aspects of the process need to be adjusted to leverage Databricks' environment and tools. This guide will cover adapting PySpark queries, optimizing performance, and designing dashboards.

**1. Setting Up Your Databricks Environment**

Databricks offers a managed Spark environment, which simplifies some of the setup steps you might be accustomed to in Jupyter Lab. Here are the steps:

1. Cluster Setup

Cluster Configuration: Databricks clusters are Spark-based and allow you to specify the instance types, storage, and configurations. You can select cluster types based on your workload (Standard, High Concurrency, or Single Node) to balance cost and performance.

Libraries and Dependencies: Any Python or Spark libraries required for your PySpark code can be installed directly on the cluster.

1. Notebook Import

You can directly import your existing Jupyter notebooks (.ipynb files) into Databricks. Databricks supports Jupyter notebooks natively, and you can run the imported notebooks with minimal modifications.

Magic Commands: Some magic commands (e.g., %pyspark, %sql) in Jupyter are slightly different in Databricks. For instance, Databricks uses %sql for SQL cells and supports %md for Markdown.

**2. Modifying PySpark Code for Databricks**

While most PySpark code runs the same way in both Jupyter Lab and Databricks, a few modifications can enhance performance and efficiency.

**A. Optimizing DataFrame Operations**

Caching: Use.cache() to store frequently accessed data in memory to avoid re-computation, which can drastically improve performance in iterative processing or interactive analysis.

Delta Lake: Databricks supports Delta Lake, an ACID-compliant storage layer that ensures reliability and speeds up data read/writes by indexing. You can convert your DataFrames to Delta tables for better performance in workflows that require updates or frequent reads.

**B. SQL Queries and User-Defined Functions (UDFs)**

Using SQL with Spark DataFrames: Databricks supports Spark SQL, which allows you to write SQL queries directly on Spark DataFrames. This is helpful if you have existing SQL queries you wish to run within Databricks.

Optimizing UDFs: Use sql.functions to create optimized user-defined functions. With Databricks are often faster than row-by-row functions.

**3. Creating Interactive Dashboards in Databricks**

Databricks has built-in tools for visualization, making it easier to create dashboards and share insights.

**Visualizing Data with Built-in Plots**

Databricks supports built-in visualization tools like bar charts, line charts, and scatter plots. You can use these tools to create quick visualizations by running SQL queries and displaying results directly in cells.

For example:

Python

Copy code display(df)

Databricks allows customizing these visuals with various chart options, enabling quick insights without external libraries.

**a. Advanced Visualizations with Libraries**

While Databricks built-in visualizations are convenient, external libraries offer more customization options. You can install these libraries on the cluster and use them to create interactive visuals.

**b. Dashboards and Alerts**

Databricks provides a dashboarding feature where you can pin visualizations to a dedicated dashboard. This is ideal for creating real-time monitoring solutions or executive dashboards.

Creating Dashboards: Simply click on the “Pin to Dashboard” option for any visualization, allowing you to organize and display multiple charts in one view.

Alerts: You can set up alerts to notify you when certain metrics go out of defined thresholds. This can be particularly useful for monitoring key indicators in production environments.

**4. Scheduling and Automation**

Databricks allows you to schedule your notebooks as jobs, automating workflows and ensuring data pipelines run at regular intervals.

Job Scheduling: You can set up a notebook as a scheduled job in Databricks, defining the frequency (e.g., daily, hourly).

Chaining Notebooks: With dbutils.notebook.run, you can create modular workflows where one notebook calls another. This helps in organizing complex workflows.

Exporting Results: If your workflow requires exporting results to dashboards or external applications, you can write results to Delta tables, JSON files, or even integrate with BI tools like Tableau.

**5. Summary**

Migrating from Jupyter Lab to Databricks can streamline your Spark-based data workflows, especially when it comes to scalability and dashboarding. While your existing PySpark code may need minimal tweaks, leveraging Databricks' built-in tools for visualization, job scheduling, and notebook parameterization can significantly enhance performance and productivity.

**5.2 importing packages**

This code imports essential PySpark libraries, enabling the creation of a SparkSession for managing Spark operations. It provides functions for data manipulation, date formatting, and aggregations, allowing column-specific operations like extracting year, quarter, month, and day of the week, summing values, and assigning unique IDs to rows

**6. Implementation in PySpark – Jupyter Lab**

**6.1 Data Ingestion and Transformation**

* Load CSV files using PySpark's spark.read.csv function and create DataFrames for each table.
* Perform necessary transformations, including renaming columns, handling missing data, and ensuring data consistency.



**6.2 Dimension Table Creation**

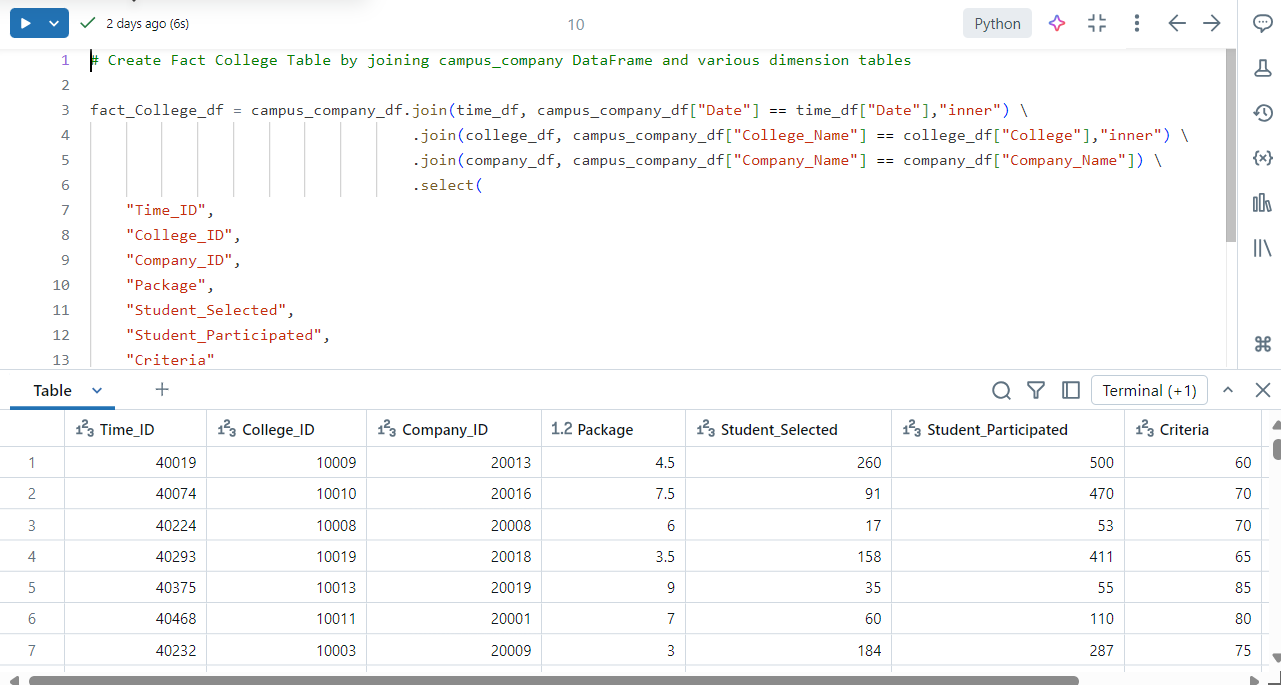
Each dimension table will be created based on the corresponding source data, ensuring unique IDs for each row in the table.

Example for creating **College Dimension**:



**6.3 Fact Table Creation**

* Join dimension tables with the source data to form fact tables.
* Example for creating the **Fact College Details** table:



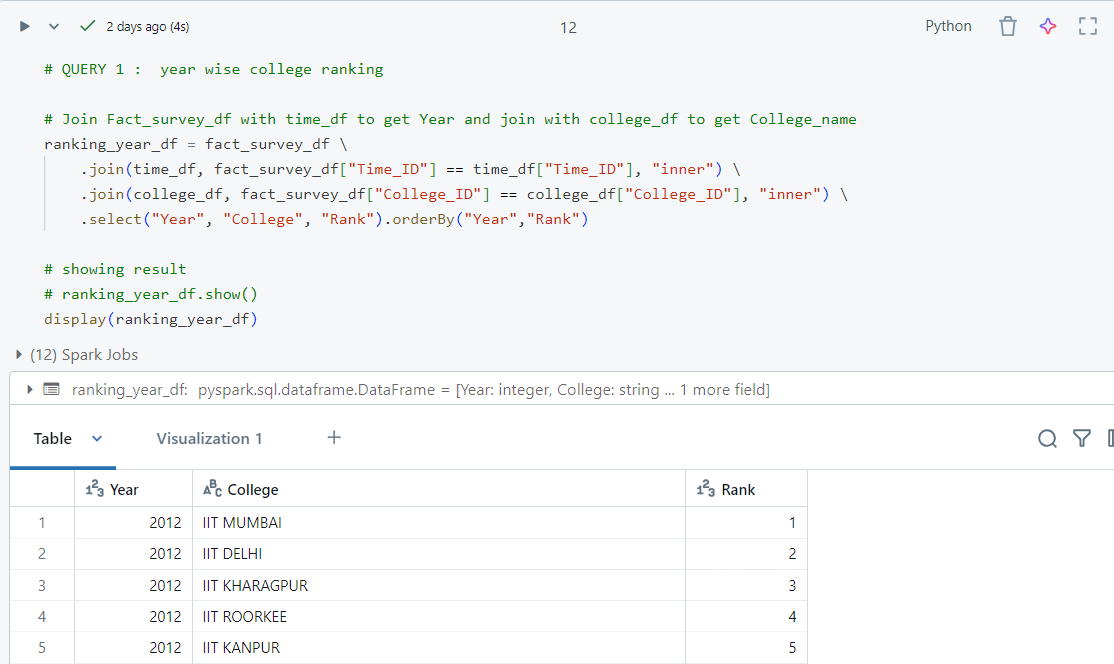
**6.4 Data Population Process**

* First, populate dimension tables, ensuring uniqueness of ID columns.
* Then, populate the fact tables using the IDs from the respective dimension tables.

**6.5 Report Generation**

Generate reports based on the aggregated data in the fact tables:

* Use PySpark's groupBy(), agg(), and rank() functions for aggregations and sorting.



**7. Reports to be Built**

**7.1 Year Wise College Ranking**

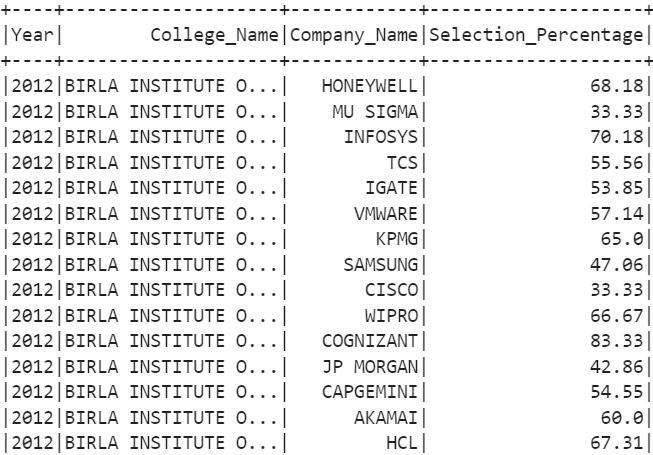
This report ranks colleges by the total number of students selected each year.

A close-up of a document

Description automatically generated

**7.2 Year Wise College Wise Percentage Selection**

This report calculates the percentage of students selected out of those who participated in the recruitment process.



**7.3 Year Wise Quarter Wise Placement**

This report will analyze the placement data quarterly, showing trends across multiple years.

A screenshot of a computer

Description automatically generated

**7.4 Total Students Selected per Company**

This report will show how many students were selected by each company across all years.

A close-up of a list

Description automatically generated

**7.5 College Ranking Year Wise**

This report ranks the colleges year-wise based on their performance (students selected).

A white paper with black text

Description automatically generated

**8. Challenges and Solutions**

**8.1 Challenges Faced**

* **Data Quality**: Some datasets had missing or inconsistent data.
* **Data Volume**: Large datasets required optimization to ensure performance.

**8.2 Solutions and Workarounds**

* Used PySpark's fillna() for missing values and dropna() for unwanted rows.
* Optimized transformations by repartitioning data and using broadcast() for small tables during joins.

**9. Conclusion and Future Work**

**Conclusion**

This mini-project successfully demonstrated the process of **Statistical Analysis of Campus Recruitments** using PySpark. The system processed large datasets, created meaningful reports, and provided insights into recruitment trends and college performance.

**Future Work**

* **Optimization**: Further optimize the system for large-scale datasets.
* **Visualization**: Add data visualization using tools.
* **Real-Time Processing**: Implement real-time recruitment data processing.

**10. Appendix (Source Files and Code)**

**10.1 Source File Overview**

* **Campus\_Source1\_Company.csv**: Contains recruitment-related data.
* **Campus\_Source2\_College.csv**: Contains survey data of participating colleges.

A close-up of a pencil

Description automatically generated

**10.2 PySpark Code for Data Transformation**

A white background with colorful text

Description automatically generated with medium confidence

**10.3 Output Files and Sample Reports**

* YearWise College Ranking.csv
* College Wise Percentage Selection.csv

A close-up of a computer screen

Description automatically generatedA computer screen shot of a computer

Description automatically generated

