University of Sheffield

INFORMATION SCHOOL

INF6032

POSTGRADUATE COURSEWORK

“BIG DATA ANALYTICS”

REGISTRATION NO. – 220201919

WORD COUNT – 3249

Description of Required Setup

Knowing more about the employed datasets is crucial for giving the needed configuration. For the sake of analysis and query resolution, I employed 2 datasets. The first is a dataset of job descriptions (JDs), which is downloaded as a CSV file from https://github.com/duyet/1skill2vec-dataset. Each row begins with a Job Description identifier and then lists the skills taken from the JD and organised into different columns. The O*NET database contains the second file. This database includes descriptions of several jobs and the required skills, grouping them under hierarchically organised headings like "Hot Technologies" or "In Demand" talents. It is used to access the "Technology Skills" file, this file is a .txt file and this file is located at the onet centre's database at - https://www.onetcenter.org/dl_files/database/db_27_2_text/Technology%20Skills.

This Assignment's work is implemented using PySpark API (Application Programming Interface) for Apache Spark, an open-source distributed computing framework. The following steps are outlined to set up a workspace using PySpark with the Skill2Vec50K dataset and the O*NET database for performing Big Data processing and analytics.

To begin, Set up a Databricks account (). As per the directions from the faculty Databricks Community Edition is used to implement the coursework. Databricks provides a collaborative environment for data science and analytics projects. Once the workspace is set up, creating a new cluster is next. The created cluster has enough resources to handle the datasets and the computations required for the analysis. This ensures that we can efficiently process and analyse large volumes of data. After creating the cluster new Python notebook, "Coursework 50K", is created. The notebook serves as the platform for executing commands and writing code. A notebook can be made for Python, SQL etc. In this coursework python notebook is created, and all the commands are based on Python programming language and its associated libraries. Next, importing the necessary library "pyspark.sql" to establish a PySpark session is done. This library provides the tools and functions required to work with Spark's structured and semi-structured data. Finally, the Skill2Vec50K and ONET datasets are uploaded to the workspace.
The data is uploaded to a specific file path, such as "/FileStore/tables/". The Skill2Vec50K dataset is uploaded as "/FileStore/tables/skill2vec_50K.csv.gz", and the ONET dataset is uploaded as "/FileStore/tables/Technology_Skills.txt". Uploading the data allows us to access and process it within the notebook. Finally, the establishment of a Connection to Data Sources is done. Skill2Vec 50K and O*NET datasets are connected in a notebook using PySpark's data source API. This allows data access from the desired file paths and performs operations using PySpark. With the environment set up, we can perform the necessary data processing and analysis tasks based on the assignment requirements. This includes data cleaning, merging or joining datasets, applying algorithms or models, generating insights and statistical analysis. Coding and execution of commands in the notebook are done to manipulate and analyse the data, leveraging the power of PySpark's distributed computing capabilities.

Data Pre-Processing Technique Implemented

In data analysis, cleaning and preparing the data is an essential step. In addition, it is crucial to fix any inconsistencies, missing numbers, duplication, or formatting problems in the datasets to provide accurate and helpful analysis.

Firstly, Importing Required Libraries and creating of spark session -

Cmd 1

```
1  #Importing Library
2  import pyspark
3  from pyspark.sql import SparkSession
4
5  #Creation of SparkSession
6  spark = SparkSession.builder.getOrCreate()
```

```
Command took 0.07 seconds -- by kgupta3@sheffield.ac.uk at 18/05/2023, 02:45:55 on My Cluster
```

To begin, importing necessary libraries enables the process and analysis of data using PySpark. One essential library is PySpark.sql, which provides powerful tools for handling structured data. By importing, we gain access to functions and methods specifically designed for working with structured datasets, allowing us to perform various data processing and analysis tasks.

After uploading the data in the file path, the following command, line 1, displays the file path.

Cmd 2

```
1  dbutils.fs.ls('dbfs:/FileStore/tables/')
2  dbutils.fs.cp('dbfs:/FileStore/tables/skill2vec_50K_csv.gz','dbfs:/FileStore/tables/skill2vec_50K.csv.gz)
3
4  # These Commands are used to check the filepath
5
```

```
Out[45]: [FileInfo(path='dbfs:/FileStore/tables/Technology_Skills.txt', name='Technology_Skills.txt', size=2513724, modificationTime=1682794759000),
 FileInfo(path='dbfs:/FileStore/tables/skill2vec_10K.csv', name='skill2vec_10K.csv', size=10731364, modificationTime=1683753380000),
 FileInfo(path='dbfs:/FileStore/tables/skill2vec_1K-1.csv', name='skill2vec_1K-1.csv', size=1074156, modificationTime=1683754063000),
 FileInfo(path='dbfs:/FileStore/tables/skill2vec_1K-2.csv', name='skill2vec_1K-2.csv', size=1074156, modificationTime=1683754195000),
 FileInfo(path='dbfs:/FileStore/tables/skill2vec_1K.csv', name='skill2vec_1K.csv', size=1074156, modificationTime=1683753844000),
 FileInfo(path='dbfs:/FileStore/tables/skill2vec_50K.csv.gz', name='skill2vec_50K.csv.gz', size=3221283, modificationTime=1684269789000),
 FileInfo(path='dbfs:/FileStore/tables/skill2vec_50K_csv.gz', name='skill2vec_50K_csv.gz', size=3221283, modificationTime=1682794748000)]
```

```
Command took 0.46 seconds -- by kgupta3@sheffield.ac.uk at 18/05/2023, 03:49:27 on My Cluster
```

Whereas line 2 is used to copy “skill2vec_50k_csv.gz” to “skill2vec_50k.csv.gz”. This is done because after uploading the data in the file path, by default, it was saved as the file name as “skill2vec_50k_csv.gz”, which as per the direction, it was an inappropriate file path and was further resolved using “dbutils”.
Next, we load the Skill2Vec50K and O*NET datasets into PySpark Dataframes. This step involves importing the datasets into our Python environment and structuring them into tabular formats that can be easily manipulated and analysed using PySpark.

Cmd 3

```
#Uploading dataset from database

filepath = "dbfs:/FileStore/tables/"

# To read the CSV file into a PySpark DataFrame
skill2vec_50K = spark.read.format("csv").option("header", "false").load(filepath+"skill2vec_50K.csv.gz")

# To read the TSV file into a PySpark DataFrame
O_NET_Technology_Skills = spark.read.option("delimiter", "\t").csv(filepath+"Technology_Skills.txt", header=True, inferSchema=True)
```

▸ (3) Spark Jobs

▸ skill2vec_50K: pyspark.sql.dataframe.DataFrame = [_c0: string, _c1: string ... 959 more fields]

▸ O_NET_Technology_Skills: pyspark.sql.dataframe.DataFrame = [O*NET-SOC Code: string, Example: string ... 4 more fields]

Command took 2.45 seconds -- by kgupta3@sheffield.ac.uk at 18/05/2023, 04:21:59 on My Cluster

After uploading both .csv and .txt files in the workspace, Data exploration, cleaning and preparation are done. This includes handling missing values, standardising data formats, removing duplicates, and transforming data for further analysis.

Cmd 4

```
#To check the rows of both the datasets

def count_it(tbl, lbl):
    print('{0:15s}: {1:,} rows'.format(lbl, tbl.count()))
count_it(skill2vec_50K, 'skill2vec_50K')
count_it(O_NET_Technology_Skills, 'O_NET_Technology_Skills')
```

▸ (4) Spark Jobs

```
skill2vec_50K  : 50,000 rows
O_NET_Technology_Skills: 31,461 rows
```

Command took 4.86 seconds -- by kgupta3@sheffield.ac.uk at 19/05/2023, 02:06:44 on My Cluster

In the Skill2vec_50 dataset, 50,000 rows were identified with 960 columns. There is no header in the file, which is mainly made up of blank spaces. Initially, the dataset was examined for any missing values. Every Job Description contains several skills; when the dataset was explored in a CSV file via MS Excel, it showed that the JD identifier with utmost skill comprises 960 columns and where not every JD had the same no. of skills, so after uploading the file in Dataframes and calling out the column, it considered the empty cells a null value. For null values, rows and columns with missing values were not dropped but were treated differently. Column 0 (C0) is kept untouched as it contains the Job Description identifier code, which is unique in nature. Column 1 to 960 is called in a loop to join each skill into one column named Skills_Required (961st column), and the Skills in Skill_Required column is converted to an array. Afterwards, C0 and Skills_Required are saved under the new Dataframe JD_identifier.

Furthermore, C0 is renamed as “Job_Description”. Dataframe JD_identifier now contains 2 columns and 50,000 rows. In the Skill_Required column, different null values are filtered, and this filtered null value dataset is stored as JD_identifier1.

View 125% Zoom Add Category Pivot Table Insert Table Chart Text Shape Media Comment Collaborate Format Organise

Sheet 1

125720	HR Executive	screening	selection	Interview	HR	Recruiter	IT F
112708	Special Teacher	Teaching	Education				
115226	consulting	fresher	IT helpdesk	Techincal Troubleshooting	international voice	international BPO	tech
19805	diploma	machining	cnc m	mould	conventional machines	die making	kno
80208	Compensation	Benefits	HR Functions	Alm	Payroll	ESS	Cor
64086	Storage Administrator						
48468	HR Operations	Exit Formalities	Shortlisting	Screening	Interviewing	Verbal Communication	End
122729	Simulink	stateflow	Matlab developer	targetlink	matlab programmer	simulink developer	mat
36721	development	information technology	api	business intelligence	problem solving	quality assurance	soa
9342	software_development	product_development_life-cycle	pdlc	systems_development_life_cycle	sdlc	development_manager	
78148	Tableau	Analytics	Financial regulation	compliance	Business Intelligence	Microstrategy	Cog
31313	Investment Banking	Secretarial Activities	Accounting	Business Finance	Company Secretary	Auditing	Taxa
123378	sap sd						
3574	factset	portfolio_management	it_portfolio_management	pmo	project_portfolio_management	charles_river	ppn
64830	Import	Export	Freight	Pune	Sanaswadi	Logistics	
4772	gis	analysis	geographic_information_system	esri	arcgis_server	arcobjects	arcs
44923	Full Stack Developer	AngularJS	SaaS application	Full Stack	Java	JavaScript	HTM
85085	Production Merchandiser Woven Nift & Pearl						
29151	handling	articles	be	i e	3b2	knowledge	sup
26480	B2B Process	Outbound Domestic process	Call centre	Outbound Calling	Outbound Sales	B2B Sales	Don
78623	Java trainer	IT Faculty	Hibernet	Professor	Core Java	Technical Training	Adv
25700	Database Administration	Database Migration	Oracle DBA	Backup	Recovery	Database Maintenance	Bus
111829	windows systems	system configuration	2010	2013	Office 365 and Exchange	Systems Administrator	Mic
23855	good communication skill	experience	field	area	skills	markets	bus
88982	Mergers and acquisitions	Fund raising	Equity	Human resources management	Sales	Business development	
60634	Javascript	JQuery	Java	Web Technologies	Software Engineering	Technology	Des
99247	events	Marketing Communication	marcom	corporate communication	branding	public relations	btl
22320	Credit Management	Disbursement	Credit Processing	Audit Compliance	Operations	LAP	HOI
10588	Financial Analysis	Agri Finance	Housing Finance	Communication Skills	Structured Finance	Sales	Ban
58196	Middle East	Permanent Recruitment	Recruitment and Staffing	Recruitment	New Business Development	Executive Search	
49058	MCOM	Female	Hadapsar	Accounting	Bcom	Tally	ERF
16382	Asp.Net MVC Developer	Asp.Net MVC	Asp.Net MVC 4 & 5	Coding	Web Services	WCF	Asp

Cmd 8

```python
from pyspark.sql.functions import array, struct

# To check the coloumn in the loop from the second to the last coloumn

columns = ["_c" + str(i) for i in range(1, 961)]

# To merge the 960 columns into a new column
Skills_Required = array(([skill2vec_50K[col_name].alias(col_name) for col_name in columns]))
skill2vec_50K = skill2vec_50K.withColumn("Skills_Required", Skills_Required)

# To display the contents of the DataFrame
skill2vec_50K.show(truncate=False)
```

▸ (1) Spark Jobs

▸ skill2vec_50K: pyspark.sql.dataframe.DataFrame = [_c0: string, _c1: string ... 960 more fields]

```
ll, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null
l, null, null, null, null, null, null, null, null, null, null, null, null]
|
|26480 |B2B Process                                   |Outbound Domestic process     |Call centre                   |Outbound Calling
|B2B Sales            |Domestic sales            |Telemarketing          |Telecaller              |Telesales              |Voice Process     |BPO
ll       |null         |null         |null            |null             |null          |null           |null         |null              |null
|null          |null    |null           |null             |null              |null                      |null    |null|null|null|null|null|null|null|null|nu
null|null|null|null|null|null|null|null|null|null|null|null|null|null|null|null|null|null|null|null|null|null|null|null|null|null|null|null|
l|null|null|null|null|null|null|null|null|null|null|null|null|null|null|null|null|null|null|null|null|null|null|null |null |null |null |null |nul
ll |null |null |null |null |null |null |null |null |null |null |null |null |null |null |null |null |null |null |null |null |null |null |null
l |null |null |null |null |null |null |null |null |null |null |null |null |null |null |null |null |null |null |null |null |null |null |null
```

Command took 11.49 seconds -- by kgupta3@sheffield.ac.uk at 26/05/2023, 17:45:18 on My Cluster

Cmd 8

```
# To select a particular column and create a subset table
JD_identifier = skill2vec_50K.select("_c0","Skills_Required")

#For renaming the coloumn
JD_identifier = JD_identifier.withColumnRenamed("_c0", "Job_Description")

# To show the subset table
JD_identifier.show(truncate=False)
```

(1) Spark Jobs

JD_identifier: pyspark.sql.dataframe.DataFrame = [Job_Description: string, Skills_Required: array]

```
+---------------+----------------+
|Job_Description|Skills_Required
|
+---------------+----------------+
```

Command took 2.18 seconds -- by kgupta3@sheffield.ac.uk at 19/05/2023, 02:06:44 on My Cluster

Cmd 9

Python

```
import pyspark.sql.functions as F

#For creating a dataframe with zero null values

JD_identifier1 = JD_identifier.withColumn('Skills_Required', F.expr('filter(Skills_Required, x -> x is not null)'))

# To display the contents of the DataFrame
JD_identifier1.show(truncate=False)
```

(1) Spark Jobs

JD_identifier1: pyspark.sql.dataframe.DataFrame = [Job_Description: string, Skills_Required: array]

```
+---------------+----------------+
|Job_Description|Skills_Required
|
+---------------+----------------+
|125720         |[HR Executive, screening, selection, Interview, HR, Recruiter, IT Recruiter, Sourcing, recruitment executive, onboarding, IT Recruitment]
|
|112708         |[Special Teacher, Teaching, Education]
|
```

Command took 1.79 seconds -- by kgupta3@sheffield.ac.uk at 19/05/2023, 02:06:44 on My Cluster

Cmd 7

```
from pyspark.sql.functions import count, when, col

# Check for null values
null_counts = O_NET_Technology_Skills.select([count(when(col(c).isNull(), c)).alias(c) for c in O_NET_Technology_Skills.columns]).collect()[0]

# Display the null value counts
for column, count in null_counts.asDict().items():
    print(f"Null values in column '{column}': {count}")
```

(2) Spark Jobs

```
Null values in column 'O*NET-SOC Code': 0
Null values in column 'Example': 0
Null values in column 'Commodity Code': 0
Null values in column 'Commodity Title': 0
Null values in column 'Hot Technology': 0
Null values in column 'In Demand': 0
```

Command took 1.29 seconds -- by kgupta3@sheffield.ac.uk at 19/05/2023, 06:14:16 on My Cluster

In the O_NET_Technology_Skills dataset, 31,461 rows with 6 columns are identified. The count of null values is zero in each column. To ensure uniqueness and avoid duplicate counts in subsequent analysis, duplicate rows in the dataset were removed. This step

ensures that each skill example is represented only once, providing accurate insights and analysis. In addition, the "Example" column in the dataset underwent a cleaning process where all the text was converted to lowercase. This step made it easier to match and compare skills between the O*NET and Skill2Vec50K datasets. By performing this cleaning, we ensured that the skills from both datasets could be seamlessly matched and analysed, leading to more accurate results and effective comparisons.
Similarly, to maintain consistency and eliminate any potential duplication of skills, the values in the "Skill" column of the dataset were standardised by converting them to lowercase. This approach ensures that skills with different cases are treated equally during aggregation and analysis. By standardising the skill column, we can accurately assess the frequency and distribution of skills within the dataset, enabling more reliable insights and effective decision-making based on the data. As a result, this JD_identifier dataset and O*NET dataset is ready for further analysis.

Cmd 11

```
1  JD_identifier1 =JD_identifier1.drop_duplicates()
2  JD_identifier1.distinct().count()
3  O_NET_Technology_Skills =O_NET_Technology_Skills.drop_duplicates()
4  O_NET_Technology_Skills.distinct().count()
```

▸ (6) Spark Jobs

▸ JD_identifier1: pyspark.sql.dataframe.DataFrame = [Job_Description: string, Skills_Required: array]

▸ O_NET_Technology_Skills: pyspark.sql.dataframe.DataFrame = [O*NET-SOC Code: string, Example: string ... 4 more fields]

```
Out[84]: 31461
```

Command took 18.55 seconds -- by kgupta3@sheffield.ac.uk at 19/05/2023, 06:14:16 on My Cluster

Apart from this, several more datasets have been created according to the need of the questions. Creating a temporary view, like "JD_identifier2", is also done as it is necessary to execute SQL commands on a PySpark DataFrame in a Python notebook. It allows querying the DataFrame using SQL syntax, making complex transformations easier, leveraging SQL expertise, and promoting code reusability. Furthermore, matplotlib.pyplot visualisation library is imported to create bar graphs.

Cmd 13

```
1  #For creating dataframe into table so that sql queries can also run
2
3  JD_identifier1.createOrReplaceTempView("JD_identifier2")
4  spark.sql("SELECT * FROM JD_identifier2").show()
```

▸ (2) Spark Jobs

```
+---------------+--------------------+
|Job_Description|     Skills_Required|
+---------------+--------------------+
|          78148|[Tableau, Analyti...|
|           4772|[gis, analysis, g...|
|           9342|[software_develop...|
|          64086|[Storage Administ...|
|         115226|[consulting, fres...|
|          31313|[Investment Banki...|
|          26480|[B2B Process, Out...|
|          85085|[Production Merch...|
|           3574|[factset, portfol...|
|          78623|[Java trainer, IT...|
|         123378|            [sap sd]|
|          29151|[handling, articl...|
|          19805|[diploma, machini...|
|          80208|[Compensation, Be...|
|         122729|[Simulink, statef...|
|         112708|[Special Teacher,...|
```

Command took 1.40 seconds -- by kgupta3@sheffield.ac.uk at 19/05/2023, 06:14:16 on My Cluster

Problem Answers

Q.1) Programmatically confirm that the number of job descriptions is as expected.

For performing the analysis in providing the results in Answer 1, several assumptions were made to ensure the validity and reliability of the results. First, the skill2vec_50K dataset containing job descriptions was assumed to provide accurate and reliable information. It was further assumed that the dataset contained no duplicate entries, as duplicates could skew the analysis and lead to inaccurate conclusions. Data cleaning techniques, such as removing duplicates and ensuring data quality, were implemented to address these assumptions using PySpark's Dataframe operations and functions. Another assumption was that the skill2vec 50K dataset was complete and representative of various job descriptions. This assumption was crucial for drawing meaningful insights and making informed decisions based on the analysis. To validate this assumption, exploratory data analysis (EDA) techniques were employed in PySpark, such as examining the distribution of job descriptions across different categories is also done further in the report. This helped assess the dataset's diversity and its ability to provide comprehensive insights into the job market.

The implementation involved the following steps, Loading the skill2vec_50K dataset into a PySpark Dataframe, Checking the number of distinct job descriptions in the dataset using PySpark's .distinct().count() function and as well from SQL query command for further exploration and comparing the count of distinct job descriptions with the expected count of 50,000 to confirm if they matched programmatically.

***Note – Results and Implementations screenshot is shown together in one screenshot.**

The result of the analysis confirmed that the number of distinct job descriptions in the skill2vec_50K dataset was as expected. In addition, the count of distinct job descriptions matched the expected count of 50,000.

Cmd 14

```
#Answer 1

spark.sql("select count(Job_Description) AS Job_Description from JD_identifier2").show()
```

▸ (3) Spark Jobs

```
+---------------+
|Job_Description|
+---------------+
|          50000|
+---------------+
```

Command took 17.02 seconds -- by kgupta3@sheffield.ac.uk at 19/05/2023, 06:14:16 on My Cluster

Confirming the expected number of job descriptions provides confidence in the dataset's completeness and integrity. Furthermore, the skill2vec_50K dataset contains diverse job descriptions, enabling comprehensive analysis and insights. Identifying the most frequent skills mentioned in the job descriptions can also be done for further exploration. The result is also retrieved from SQL query to represent the result in tabular format.

Q.2) Work out the frequencies with which distinct skills are mentioned in job descriptions and present your report's top 10 (in order of decreasing frequency) skills alongside the frequency of each across the entire dataset.

In conducting this analysis in providing the results in Answer 2, several assumptions were made. Firstly, it was assumed that the job descriptions included in the skill2Vec_50K dataset were extracted accurately and reflected the skills required for different job roles. Secondly, it was assumed that the dataset provided a comprehensive representation of job descriptions across diverse industries, ensuring a wide range of skills were covered. Additionally, it was assumed that the dataset had undergone pre-processing and cleaning, eliminating any outside spaces and misleading information that could impact the analysis. Lastly, it was assumed that the frequency of skills mentioned in the job descriptions directly corresponded to their significance and demand in the job market, allowing us to infer the importance of specific skills based on their frequency.

To analyse the skill frequencies in the skill2Vec_50K dataset, the implementation performed the following steps using PySpark. First, we imported the necessary data processing and analysis libraries in PySpark. Then, we extracted the distinct skills mentioned in the job descriptions by exploding the Required Skill array for retrieving each skill with each JD. Then grouping of the Skill Required column is done. Next, to determine the frequency of each skill, we counted its occurrences in the dataset. The skills were then sorted in descending order based on their frequency. Finally, we selected the top 10 skills with the highest frequencies and recorded the skill names and their respective frequencies. These steps allowed us to identify the most frequently mentioned skills in the skill2Vec_50K dataset, providing valuable insights into the skill requirements in job descriptions.

Cmd 15

```
#Answer 2

spark.sql("SELECT Skills, COUNT(*) AS Frequency FROM JD_identifier2 LATERAL VIEW EXPLODE(Skills_Required) AS Skills GROUP BY Skills ORDER BY Frequency DESC LIMIT 10").show()
```

(3) Spark Jobs

```
+--------------------+---------+
|              Skills|Frequency|
+--------------------+---------+
|                Java|     1911|
|          Javascript|     1770|
|               Sales|     1705|
|Business Development|     1545|
|    Web Technologies|     1313|
|Communication Skills|     1307|
|         development|     1238|
|           Marketing|     1184|
|             Finance|     1078|
|                HTML|     1067|
+--------------------+---------+
```

Command took 24.99 seconds -- by kgupta3@sheffield.ac.uk at 19/05/2023, 06:14:16 on My Cluster

Cmd 16

```
from pyspark.sql.functions import explode

exploded_df = JD_identifier1.select(explode(JD_identifier1.Skills_Required).alias('Skill'))
exploded_df.groupby('Skill').count().sort("count",ascending = False).show()
```

(3) Spark Jobs

exploded_df: pyspark.sql.dataframe.DataFrame = [Skill: string]

```
|                Java| 1911|
|          Javascript| 1770|
|               Sales| 1705|
|Business Development| 1545|
|    Web Technologies| 1313|
|Communication Skills| 1307|
|         development| 1238|
|           Marketing| 1184|
|             Finance| 1078|
|                HTML| 1067|
|                 SQL| 1027|
|               sales|  950|
|                 CSS|  934|
|          Accounting|  923|
|              JQuery|  883|
|  Project Management|  810|
|                 BPO|  803|
|         Recruitment|  788|
|                java|  784|
|          javascript|  780|
+--------------------+-----+
```

Command took 23.58 seconds -- by kgupta3@sheffield.ac.uk at 19/05/2023, 06:14:16 on My Cluster

The result of the analysis confirmed that the top 10 distinct skills mentioned in the Job Description were as expected.

Cmd 17

```
skill_counts = exploded_df.groupby('Skill').count()
sorted_counts = skill_counts.sort("count", ascending=False)
top_skills = sorted_counts.limit(10)
top_skills.show()
```

(3) Spark Jobs

skill_counts: pyspark.sql.dataframe.DataFrame = [Skill: string, count: long]

sorted_counts: pyspark.sql.dataframe.DataFrame = [Skill: string, count: long]

top_skills: pyspark.sql.dataframe.DataFrame = [Skill: string, count: long]

Skill	count
Java	1911
Javascript	1770
Sales	1705
Business Development	1545
Web Technologies	1313
Communication Skills	1307
development	1238
Marketing	1184
Finance	1078
HTML	1067

Command took 24.63 seconds -- by kgupta3@sheffield.ac.uk at 19/05/2023, 06:14:16 on My Cluster

The analysis results provide valuable insights into the current job market's Top Skill demands, enabling job seekers to align their skills with employer preferences. Organisations can also benefit from this information by aligning their talent acquisition strategies accordingly. Additionally, the analysis was performed using SQL queries, enhancing the versatility and latency of the approach. Future explorations may involve studying skill trends, comparing industry frequencies, and identifying emerging skills. These efforts contribute to a deeper understanding of skill dynamics and support informed decision-making for job seekers and organisations.

Q.3) Find the 5 most frequent numbers of skills in JDs across the dataset.

During the analysis and presentation of the results in Answer 3, it was assumed that the PySpark functions and methods employed, such as "countDistinct," "count," and "groupBy," reliably computed the skill frequencies and yielded the desired outcomes. The assumption entailed correctly implementing these functions and aligning with the analysis's intended logic and requirements. Furthermore, it was assumed that choosing the top 5 most frequent skills offered a representative sample of the skills mentioned in the dataset. Based on their frequency, these skills indicated key skills in demand. These assumptions were crucial for interpreting the results and drawing meaningful insights from the analysis.

The implementation involved using PySpark's DataFrame operations and functions. First, the "Skills_Required" column was exploded to create a new row for each skill mentioned in a job description. Grouping was performed on the "Job_Description" column to count the distinct skills per job description using the "countDistinct" function. Further grouping

was done on the "Num_Skills" column to calculate the frequency of job descriptions having a specific number of skills using the count function. The results were then ordered in descending order of frequency using the orderBy function. The top 5 frequencies were selected using the limit function. The total frequency count was obtained for further exploration by summing the "Frequency" column using the sum function.

Cmd 18

```
#Answer 3

from pyspark.sql.functions import countDistinct , count , col , sum

freq_skills = JD_identifier1.selectExpr("Job_Description", "explode(Skills_Required) as Skill") \
                .groupBy("Job_Description") \
                .agg(countDistinct("Skill").alias("Num_Skills")) \
                .groupBy("Num_Skills") \
                .agg(count("Job_Description").alias("Frequency")) \
                .orderBy(col("Frequency").desc()) \
                .limit(5)

freq_skills.show()
freq_skills.select(sum(freq_skills.Frequency)).show()
```

▸ (10) Spark Jobs

▸ freq_skills: pyspark.sql.dataframe.DataFrame = [Num_Skills: long, Frequency: long]

```
+----------+---------+
|Num_Skills|Frequency|
+----------+---------+
|        10|    10477|
|         5|     3432|
|         6|     3405|
|         1|     3386|
|         7|     3345|
+----------+---------+

+--------------+
|sum(Frequency)|
+--------------+
|         24045|
+--------------+
```

Command took 1.05 minutes -- by kgupta3@sheffield.ac.uk at 23/05/2023, 14:18:42 on My Cluster

The Result above displays the top 5 frequencies of the number of skills in job descriptions. In addition, the analysis revealed the five most frequent numbers of skills mentioned in job descriptions across the dataset. This information can be valuable in understanding the typical skill requirements for various job roles. For Further exploration, Skills in lower case and separate skill without distinct was tested, and the result was slightly changed from the actual answer. Additionally, the sum of distinct skills was calculated to find the total distinct skills.

Cmd 19

```
#Skills in Lowercase

from pyspark.sql.functions import countDistinct , count , col , explode

freq_skills1 = JD_identifier1.selectExpr("Job_Description", "explode(Skills_Required) as Skill") \
                .selectExpr("Job_Description", "lower(Skill) as Skill") \
                .groupBy("Job_Description") \
                .agg(countDistinct("Skill").alias("Num_Skills")) \
                .groupBy("Num_Skills") \
                .agg(count("Job_Description").alias("Num_JDs")) \
                .orderBy(col("Num_JDs").desc()) \
                .limit(5)

freq_skills1.show()
```

▸ (5) Spark Jobs

▸ freq_skills1: pyspark.sql.dataframe.DataFrame = [Num_Skills: long, Num_JDs: long]

```
+----------+-------+
|Num_Skills|Num_JDs|
+----------+-------+
|        10|  10484|
|         5|   3426|
|         6|   3411|
|         1|   3386|
|         7|   3339|
+----------+-------+
```

Command took 29.60 seconds -- by kgupta3@sheffield.ac.uk at 23/05/2023, 14:18:42 on My Cluster

Cmd 20

```
#Skills without distinct count

freq_skills2 = JD_identifier1.selectExpr("Job_Description", "explode(Skills_Required) as Skill")\
                .groupBy("Job_Description") \
                .agg(count("Skill").alias("Num_Skills")) \
                .groupBy("Num_Skills") \
                .agg(count("Job_Description").alias("Frequency")) \
                .orderBy(col("Frequency").desc())


freq_skills2.show(5)
```

▸ (4) Spark Jobs

▸ freq_skills2: pyspark.sql.dataframe.DataFrame = [Num_Skills: long, Frequency: long]

```
+----------+---------+
|Num_Skills|Frequency|
+----------+---------+
|        10|    10480|
|         5|     3436|
|         6|     3408|
|         1|     3367|
|         7|     3346|
+----------+---------+
only showing top 5 rows
```

Command took 27.16 seconds -- by kgupta3@sheffield.ac.uk at 23/05/2023, 14:18:42 on My Cluster

Q.4) Check how the distribution of the frequencies with which distinct skills are mentioned in JDs changes if you lowercase all the skills.

In Answer 4, it was assumed that converting all the skills to lowercase would affect the distribution of skill frequencies in the job descriptions. The assumption was that lowercase transformation would standardise the skills and potentially merge different cases of the same skill, resulting in a change in the frequency distribution. It was also assumed that the query and functions used in Spark SQL, such as "lower" and "GROUP BY," were correctly implemented and would produce the desired results.

To investigate the impact of lowercasing the skills on their frequency distribution, the PySpark.SQL query was executed to retrieve the answer fast and for the reusability of the code. This query first converted all the skills to lowercase using the "lower" function. Then, it calculated the frequency of each distinct skill in the job descriptions by counting the occurrences. The results were grouped by the lowercase skill names and ordered in descending order of frequency. Finally, the top 10 skills with the highest frequencies were selected and displayed.

Cmd 21

```
#Answer 4

spark.sql("SELECT lower(Skills) AS skills , COUNT(*) AS Frequency FROM JD_identifier2 LATERAL VIEW EXPLODE(Skills_Required) AS Skills GROUP BY lower(Skills) ORDER BY Frequency DESC LIMIT 10").show()
```

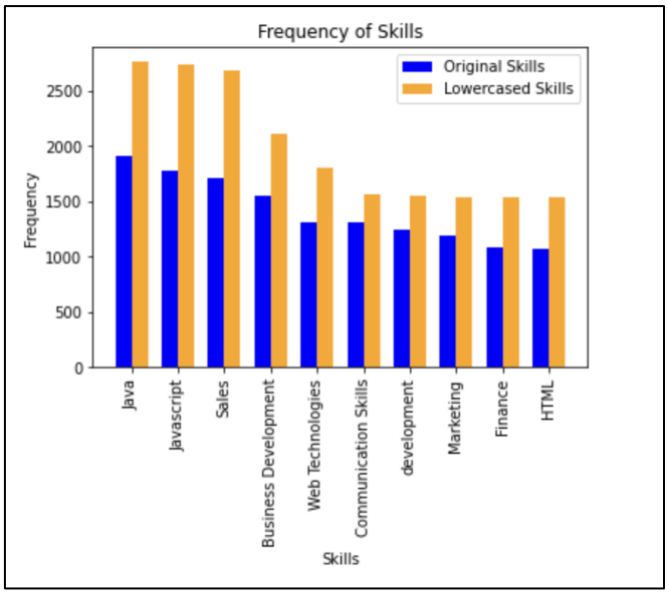
▸ (3) Spark Jobs

```
+--------------------+---------+
|              skills|Frequency|
+--------------------+---------+
|                java|     2759|
|          javascript|     2738|
|               sales|     2680|
|business development|     2108|
|           marketing|     1809|
|                 sql|     1564|
|              jquery|     1547|
|                html|     1540|
|communication skills|     1539|
|                 bpo|     1530|
+--------------------+---------+
```

Command took 25.44 seconds -- by kgupta3@sheffield.ac.uk at 20/05/2023, 03:55:46 on My Cluster

These snippets showcase the output table displaying the lowercase skills and their corresponding frequencies, representing the changed distribution of skill frequencies after lower casing. The analysis of lowercasing the skills in the job descriptions can provide insights into the impact of case sensitivity on skill frequency distribution. Converting all the skills to lowercase makes it possible to identify skills previously represented with different cases but refer to the same skill. This can help in reducing duplication and aggregating similar skills. The results obtained from this analysis can be compared with the original skill frequency distribution to assess the extent of changes.
Further explorations could involve studying the specific skills most affected by lowercasing and investigating the implications of these changes for job matching and skill-based analyses. Additionally, comparison can be done using visualisation, as depicted below. The grouped Bar graph shows the distribution of frequencies before and after lowercasing the

skills. The client perspective must analyse the data carefully as even lowercasing the skill can make a difference in the count.



Cmd 22

Python

```
import matplotlib.pyplot as plt

# Original Skills and Frequencies
original_results = spark.sql("SELECT Skills, COUNT(*) AS Frequency FROM JD_identifier2 LATERAL VIEW EXPLODE(Skills_Required) AS Skills GROUP BY Skills ORDER BY Frequency DESC LIMIT 10")
original_skills = [row.Skills for row in original_results.collect()]
original_frequencies = [row.Frequency for row in original_results.collect()]

# Lowercased Skills and Frequencies
lowercased_results = spark.sql("SELECT lower(Skills) AS Skills, COUNT(*) AS Frequency FROM JD_identifier2 LATERAL VIEW EXPLODE(Skills_Required) AS Skills GROUP BY lower(Skills) ORDER BY Frequency DESC LIMIT 10")
lowercased_skills = [row.Skills for row in lowercased_results.collect()]
lowercased_frequencies = [row.Frequency for row in lowercased_results.collect()]

# Creating figure and axis
fig, ax = plt.subplots()

# Setting width of each bar
bar_width = 0.35

# Setting positions of the bars on the x-axis
r1 = range(len(original_skills))
r2 = [x + bar_width for x in r1]

# Plotting the original frequencies
ax.bar(r1, original_frequencies, color='blue', width=bar_width, label='Original Skills')

# Plotting the lowercased frequencies
ax.bar(r2, lowercased_frequencies, color='orange', width=bar_width, label='Lowercased Skills')

# Setting labels and title
ax.set_xlabel('Skills')
ax.set_ylabel('Frequency')
ax.set_title('Frequency of Skills')
ax.set_xticks([r + bar_width/2 for r in range(len(original_skills))])
ax.set_xticklabels(original_skills, rotation='vertical')

# Adding a legend
ax.legend()

# Showing the plot
plt.show()
```

▸ (8) Spark Jobs

▸ original_results: pyspark.sql.dataframe.DataFrame = [Skills: string, Frequency: long]

▸ lowercased_results: pyspark.sql.dataframe.DataFrame = [Skills: string, Frequency: long]

The original talents and their frequencies are displayed in blue in the grouped bar graph created by this code, while the lowercase skills and their frequencies are displayed in orange. The talents are shown on the x-axis, while frequency is shown on the y-axis. Each bar has its corresponding talent labelled, and the plot has a legend to help you tell the two groups apart.

Q.5) To gain additional information about the sought-after skills, you'd like to join the (lower cased) skills from JDs with the skills listed in the Example column in the O*NET dataset. Find the change in the number of skills before and after the join.

In answer 5, several assumptions were made regarding successfully matching lower-cased skills between the JD and ONET datasets through the Inner Join operations other than Left, Right, and Full Outer Join. First, the inner join process was assumed to capture the shared skills between the two datasets accurately because the left or right join would have included all the records from one dataset (either the JD or ONET datasets). A full outer join would have included all the data from both datasets, including matching and non-matching records. Additionally, it was assumed that the lower-cased skills from JDs and the ONET dataset were matched correctly, ensuring the accuracy of the results. Furthermore, it was assumed that the counts obtained for the original skills and the skills after the join accurately represented the unique skills present in each dataset.

The job descriptions dataset underwent pre-processing to implement the analysis by lower-casing the "Skill" column and eliminating duplicate entries. Similarly, the O*NET dataset was modified by lower-casing the "Example" column. The two datasets were then joined based on matching lower-cased skills. The count of distinct skills in the lower-cased "Skill" column before the join provided the number of original skills. Subsequently, the total row count in the resulting joined dataset determined the number of skills after the join. This implementation allowed for comparing and identifying skills in both datasets, providing valuable insights into their overlap and potential alignment.

Cmd 22

```
#Answer 5

from pyspark.sql.functions import lower , length

jd_lower1 = JD_identifier1.selectExpr("Job_Description", "explode(Skills_Required) as Skill")
jd_lower1 = jd_lower1.drop_duplicates()
jd_lower2 = jd_lower1.selectExpr("Job_Description", "lower(Skill) as Skill")
onet_lower1 = O_NET_Technology_Skills.withColumn("Example",lower(col("Example")))
joinedDF1 = jd_lower2.join(onet_lower1, onet_lower1['Example'] == jd_lower2["Skill"],'inner')
print("Original number of skills before join:", jd_lower2.count())
print("Skill after join:", joinedDF1.count())
```

```
▸ (10) Spark Jobs
▸ jd_lower1: pyspark.sql.dataframe.DataFrame = [Job_Description: string, Skill: string]
▸ jd_lower2: pyspark.sql.dataframe.DataFrame = [Job_Description: string, Skill: string]
▸ onet_lower1: pyspark.sql.dataframe.DataFrame = [O*NET-SOC Code: string, Example: string ... 4 more fields]
▸ joinedDF1: pyspark.sql.dataframe.DataFrame = [Job_Description: string, Skill: string ... 6 more fields]
Original number of skills before join: 463803
Skill after join: 1101498
Command took 1.01 minutes -- by kgupta3@sheffield.ac.uk at 26/05/2023, 17:45:18 on My Cluster
```

Result: The number of original skills before joining: 463803
The number of skills after the join: 1101498

By merging the Job Descriptions dataset and the ONET dataset, we could effectively identify shared skills. Furthermore, we can confidently determine the degree of compatibility between the datasets by analysing the skill counts pre and post-join. For

Further Exploration, this question was implemented using Spark's RDD Map function without removing the duplicates. Another exploration was done after removing the duplicates from the datasets after converting the skills into lowercase and joining both datasets. But both explorations showed different results. Spark's RDD Map function showed Low Latency in the execution of the code.

Cmd 24

```
from pyspark.sql.functions import lower , length

df_jd_lower = JD_identifier1.selectExpr("Job_Description", "explode(Skills_Required) as Skill") \
                            .selectExpr("Job_Description", "lower(Skill) as Skill")
df_onet_lower = O_NET_Technology_Skills.select(lower("Example").alias("Example"))

joined_df = df_jd_lower.join(df_onet_lower, df_jd_lower.Skill.contains(df_onet_lower.Example), "inner")

joinedDF = df_jd_lower.join(df_onet_lower, df_onet_lower['Example'] == df_jd_lower["Skill"],'inner')

num_orig_skills = JD_identifier1.selectExpr("size(Skills_Required)").rdd.map(lambda x: x[0]).sum()

print("Before Join:", num_orig_skills)
print("After Join:",joinedDF.count())
```

```
▸ (7) Spark Jobs
▸ df_jd_lower: pyspark.sql.dataframe.DataFrame = [Job_Description: string, Skill: string]
▸ df_onet_lower: pyspark.sql.dataframe.DataFrame = [Example: string]
▸ joined_df: pyspark.sql.dataframe.DataFrame = [Job_Description: string, Skill: string ... 1 more field]
▸ joinedDF: pyspark.sql.dataframe.DataFrame = [Job_Description: string, Skill: string ... 1 more field]
Before Join: 463908
After Join: 1101742
Command took 51.04 seconds -- by kgupta3@sheffield.ac.uk at 26/05/2023, 17:45:18 on My Cluster
```

Cmd 25

```
from pyspark.sql.functions import lower , length

df_jd_lower11 = JD_identifier1.selectExpr("Job_Description", "explode(Skills_Required) as Skill")
df_jd_lower22 = df_jd_lower11.selectExpr("Job_Description", "lower(Skill) as Skill")

df_jd_lower22 = df_jd_lower22.drop_duplicates()

df_onet_lower11 = O_NET_Technology_Skills.withColumn("Example",lower(col("Example")))

df_onet_lower11 = df_onet_lower11.drop_duplicates()

joinedDF11 = df_jd_lower22.join(df_onet_lower11, df_onet_lower11['Example'] == df_jd_lower22["Skill"],'inner')
joinedDF11 = joinedDF11.drop_duplicates()



print("Before Join:", df_jd_lower22.count())
print("After Join:" , joinedDF11.count())
```

```
▸ (9) Spark Jobs
▸ df_jd_lower11: pyspark.sql.dataframe.DataFrame = [Job_Description: string, Skill: string]
▸ df_jd_lower22: pyspark.sql.dataframe.DataFrame = [Job_Description: string, Skill: string]
▸ df_onet_lower11: pyspark.sql.dataframe.DataFrame = [O*NET-SOC Code: string, Example: string ... 4 more fields]
▸ joinedDF11: pyspark.sql.dataframe.DataFrame = [Job_Description: string, Skill: string ... 6 more fields]
Before Join: 463517
After Join: 1100214
Command took 1.07 minutes -- by kgupta3@sheffield.ac.uk at 26/05/2023, 17:45:19 on My Cluster
```

Additionally, investigating the unmatched skills could provide insights into unique or industry-specific skills that are not captured in the O*NET dataset.

Q.6) Find the 10 most frequent “Commodity Titles” across all the job descriptions.

In this analysis, we made several assumptions to ensure the accuracy and feasibility of the results. Firstly, we assumed that the "Commodity Title" column is accessible after performing the join operation between the job descriptions (JDs) and O*NET datasets. This assumption is crucial because it enables us to identify the category or type associated with each job description. Secondly, we assumed that the "Commodity Title" represents a meaningful classification or grouping of job descriptions. By considering the "Commodity Title" as a category, we can analyse the frequency of occurrence for each skill and determine the most common ones. Lastly, all the assumptions and analyses were performed on the joined dataset, combining the relevant information from the JDs and O*NET datasets. This ensures that we work with a consolidated dataset that provides comprehensive insights into the skills and "Commodity Title" values associated with job descriptions.

The joined dataset from the previous step was utilised to extract the 10 most frequent "Commodity Title" values across all job descriptions. The joined dataset contained information from the job descriptions and the O*NET dataset, enabling access to the "Commodity Title" column. The "Commodity Title" column grouped the dataset, and the count of occurrences for each title was calculated using the count() function. The resulting dataset was sorted in descending order based on the count values. To identify the top 10 most frequent "Commodity Title" values, the sorted dataset showed only the first 10 rows. The results were displayed using the show() function, and the count values were displayed in descending order.

Cmd 23

```
#Ans 6

joinedDF2 = joinedDF1.groupBy("Commodity Title")\
                     .count()\
                     .sort('count',ascending=False)

joinedDF2.show(10,False)
```

▸ (6) Spark Jobs

▸ joinedDF2: pyspark.sql.dataframe.DataFrame = [Commodity Title: string, count: long]

```
+-------------------------------------------------+------+
|Commodity Title                                  |count |
+-------------------------------------------------+------+
|Object or component oriented development software|324521|
|Web platform development software                |298754|
|Operating system software                        |190926|
|Development environment software                 |53013 |
|Data base management system software             |44132 |
|Analytical or scientific software                |33552 |
|Web page creation and editing software           |31682 |
|Data base user interface and query software      |29436 |
|Spreadsheet software                             |18568 |
|File versioning software                         |13846 |
+-------------------------------------------------+------+
only showing top 10 rows
```

Command took 28.54 seconds -- by kgupta3@sheffield.ac.uk at 21/05/2023, 03:08:45 on My Cluster

For further exploration, the Top 10 Frequent Commodity Titles are shown in Bar Graph to enhance understanding of skill frequency and patterns.

Cmd 27

```
#Graphical representation of answer 6

import matplotlib.pyplot as plt
titles = joinedDF2.select("Commodity Title").limit(10).rdd.flatMap(lambda x: x).collect()
counts = joinedDF2.select("Count").limit(10).rdd.flatMap(lambda x: x).collect()
plt.bar(titles, counts)
plt.xticks(rotation=90)
plt.xlabel("Commodity Title")
plt.ylabel("Count")
plt.title("Top 10 Frequent Commodity Titles")
plt.show()
```