Big Data Management
Griffith College Dublin

Assignment 2

Rohin Mehra
Enrolment number: 3082862
MSc.Big Data Analysis & Management
17 April 2023

Abstract

This analysis uses the CRAN package download logs dataset to answer several questions related to R package downloads, popularity, and usage patterns. The dataset consists of the following fields:

date

time

size

r_version

r_arch

r_os

package

version

country

ip_id

We use Apache Spark to process the data and perform various analytics tasks using the RDD API.

We will make use of two Python libraries pyspark.sql and datetime. The pyspark.sql library is a part of Apache Spark, a big data processing framework that allows distributed processing of large datasets across clusters of computers. It provides APIs for working with structured and semi-structured data using DataFrame and SQL-like queries.

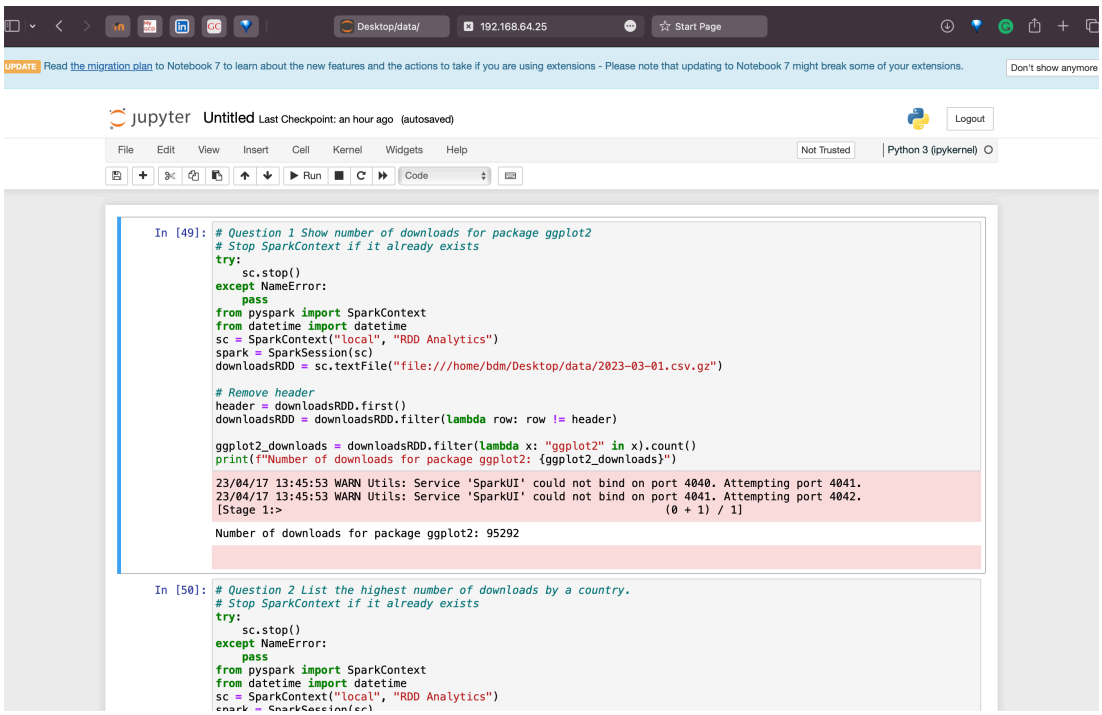
The datetime library is a part of the Python Standard Library and provides classes and methods for working with dates and times. It allows the manipulation of dates, times, and timestamps and also includes time zone support.

Question Wise Explanation

1. Number of downloads for package ggplot2

The code filters the RDD for records containing the ggplot2 package and counts the number of occurrences. This gives us the total number of downloads for the ggplot2 package.

1. We first stop any existing SparkContext to avoid conflicts.
2. We import the necessary libraries and create a new SparkContext named "RDD Analytics".
3. We read the dataset from the given file path using sc.textFile().
4. We remove the header from the dataset to exclude it from our analysis.
5. We filter the dataset for rows containing the package name "ggplot2" and count the number of rows that match this condition.
6. We print the total number of downloads for the ggplot2 package.

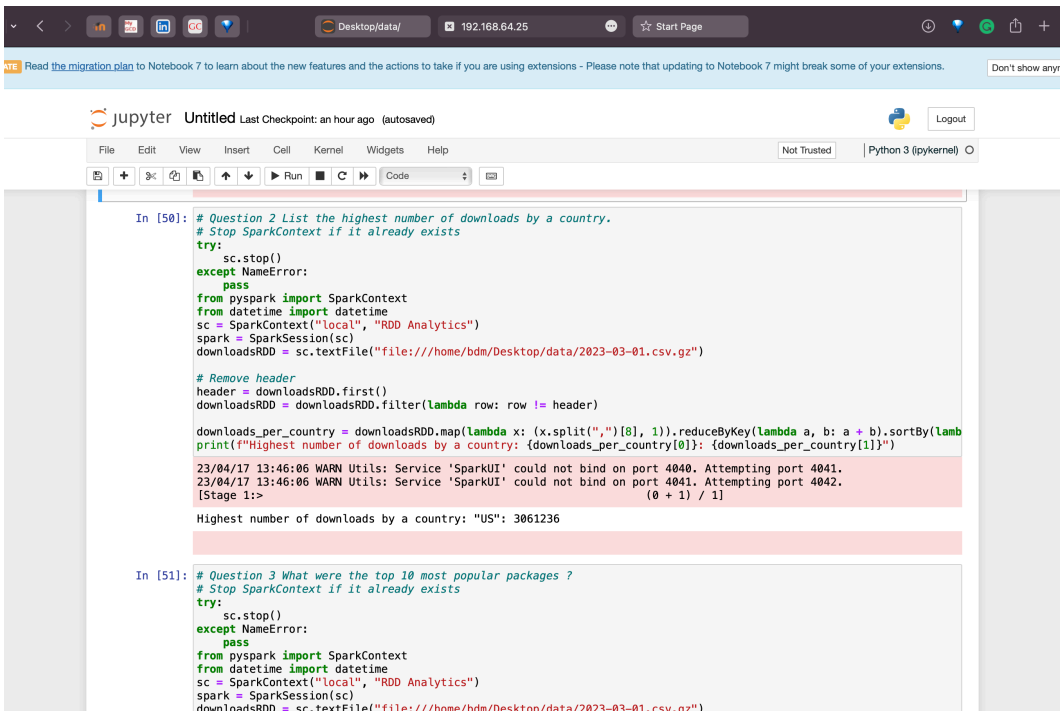


Question 1 code and output

2. Highest number of downloads by a country

For this analysis, we create a key-value pair RDD with the country as the key and the value set to 1. We then reduce this RDD by key, adding up the values for each country, and finally sorting the reduced RDD by count. The first record provides the country with the highest number of downloads.

1. We first stop any existing SparkContext to avoid conflicts.
2. We import the necessary libraries and create a new SparkContext named "RDD Analytics".
3. We read the dataset from the given file path using sc.textFile().
4. We remove the header from the dataset to exclude it from our analysis.
5. We create a key-value pair RDD where the key is the country and the value is 1. Then, we use reduceByKey() to sum the values for each country.



Question 2 code and output

6. We sort the result in descending order of the download count and retrieve the first entry, which corresponds to the country with the highest number of downloads.

7. We print the country with the highest number of downloads and its download count.

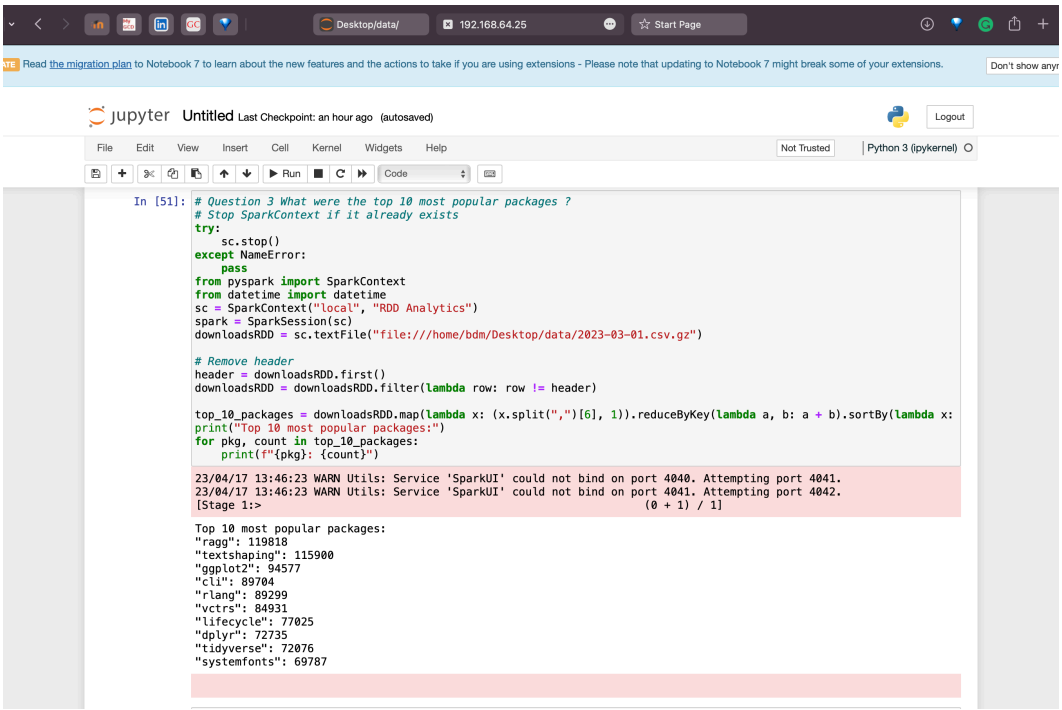
3. Top 10 most popular packages

To identify the top 10 most popular packages, we create a key-value pair RDD with the package name as the key and the value set to 1. We reduce this RDD by key, adding up the values for each package, and finally sorting the reduced RDD by count. We then take the top 10 records to get the most popular packages.

1. We first stop any existing SparkContext to avoid conflicts.

2. We import the necessary libraries and create a new SparkContext named "RDD Analytics".

3. We read the dataset from the given file path using sc.textFile().



Question 3 code and output

4. We remove the header from the dataset to exclude it from our analysis.

5. We create a key-value pair RDD where the key is the package name and the value is 1. Then, we use reduceByKey() to sum the values for each package.

6. We sort the result in descending order of the download count and retrieve the top 10 entries, which correspond to the most popular packages.

7. We print the top 10 most popular packages and their download counts.

4. Most popular package in Ireland

We filter the RDD for records from Ireland, then create a key-value pair RDD with the package name as the key and the value set to 1. We reduce this RDD by key, adding up the values for each package, and finally sorting the reduced RDD by count. The first record provides the most popular package in Ireland.

1. We first stop any existing SparkContext to avoid conflicts.

2. We import the necessary libraries and create a new SparkContext named "RDD Analytics".

3. We read the dataset from the given file path using sc.textFile().

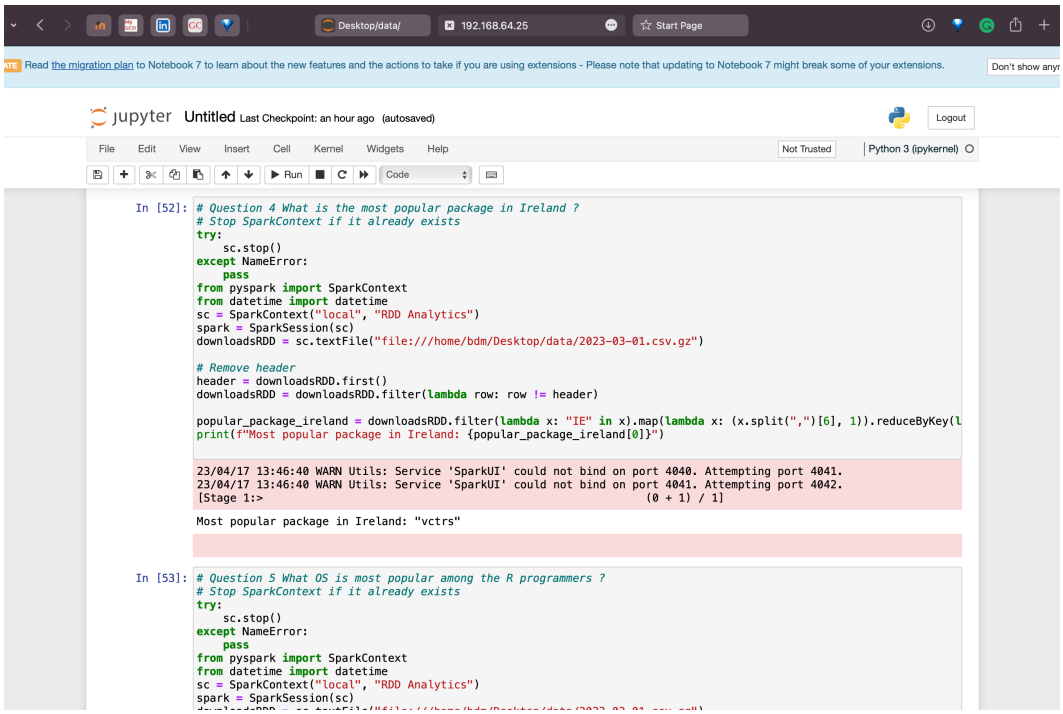
4. We remove the header from the dataset to exclude it from our analysis.

5. We filter the dataset to include only records from Ireland using the filter() function.

6. We create a key-value pair RDD where the key is the package name and the value is 1. Then, we use reduceByKey() to sum the values for each package.

7. We sort the result in descending order of the download count and retrieve the first entry, which corresponds to the most popular package in Ireland.

8. We print the most popular package in Ireland.



Question4 code and output

5. Most popular OS among R programmers

To determine the most popular OS among R programmers, we create a key-value pair RDD with the OS as the key and the value set to 1. We reduce this RDD by key, adding up the values for each OS, and finally sorting the reduced RDD by count. The first record provides the most popular OS.

1. We first stop any existing SparkContext to avoid conflicts.

2. We import the necessary libraries and create a new SparkContext named "RDD Analytics".

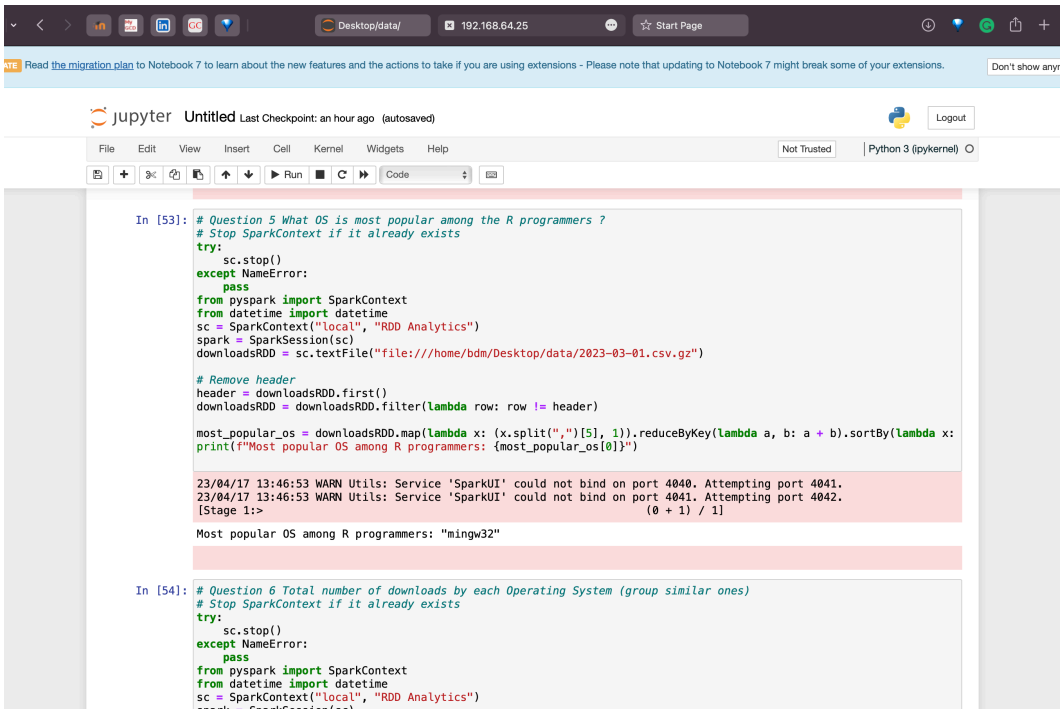
3. We read the dataset from the given file path using sc.textFile().

4. We remove the header from the dataset to exclude it from our analysis.

5. We create a key-value pair RDD where the key is the OS name and the value is 1. Then, we use reduceByKey() to sum the values for each OS.

6. We sort the result in descending order of the download count and retrieve the first entry, which corresponds to the most popular OS among R programmers.

7. We print the most popular OS among R programmers.

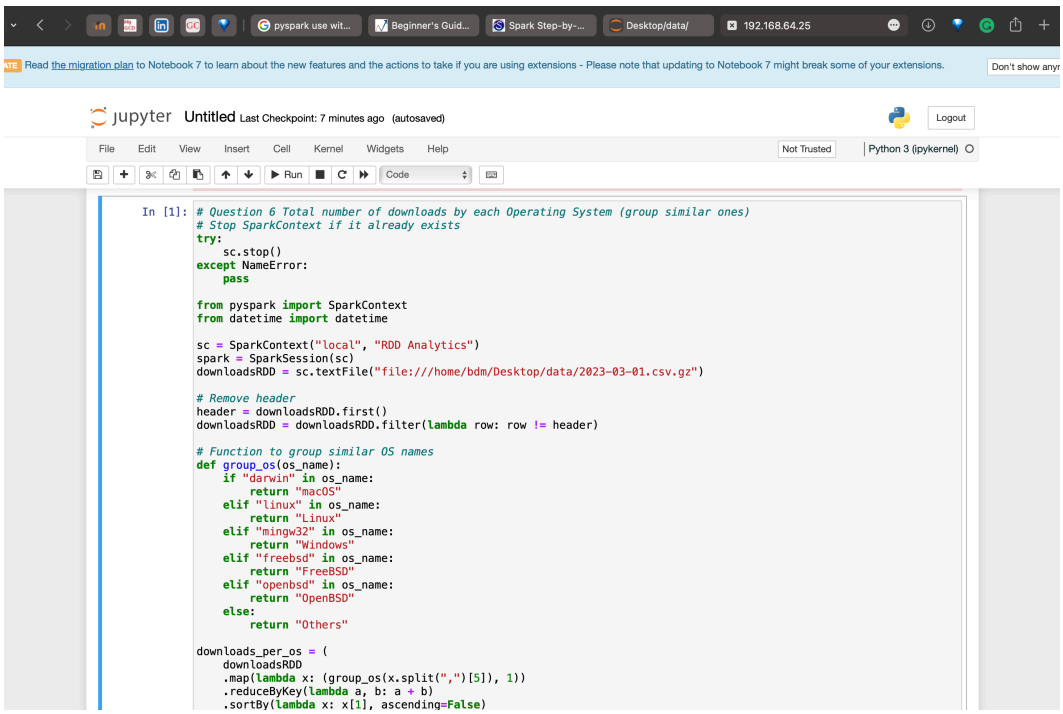


Question 5 code and output

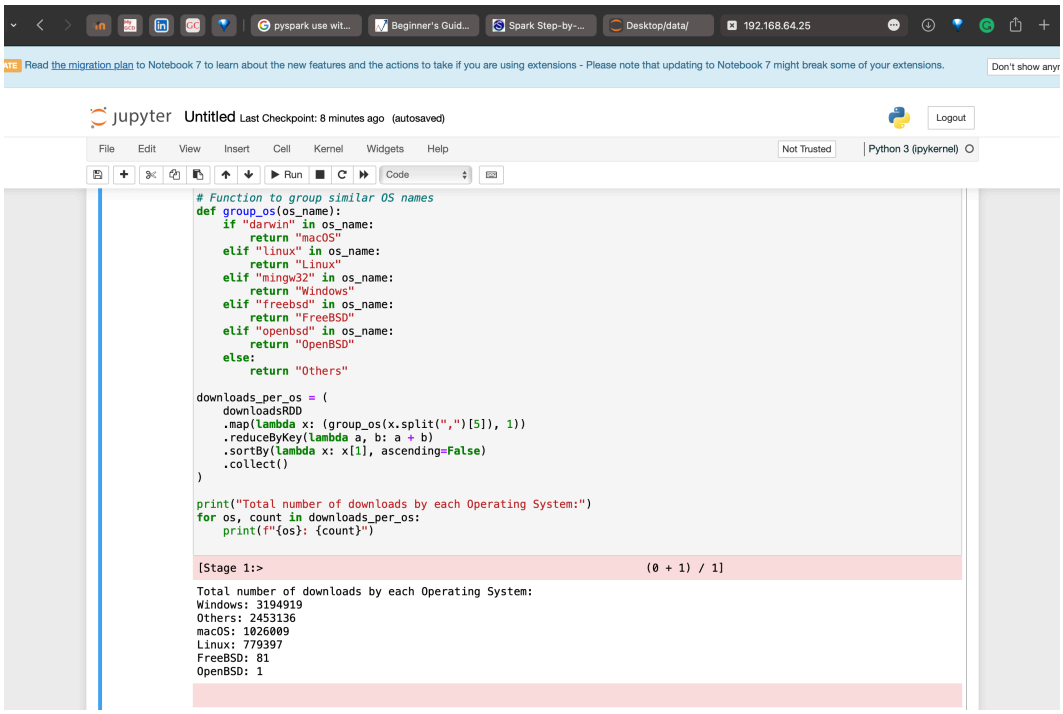
6. Total number of downloads by each Operating System

We use a similar approach as in the previous analysis, but instead of selecting the top OS, we collect all the records to provide the total number of downloads for each OS.

1. We first stop any existing SparkContext to avoid conflicts.
2. We import the necessary libraries and create a new SparkContext named "RDD Analytics".
3. We read the dataset from the given file path using sc.textFile().
4. We remove the header from the dataset to exclude it from our analysis.
5. We define a function group_os() that groups similar OS names.
6. We create a key-value pair RDD where the key is the grouped OS name and the value is 1. Then, we use reduceByKey() to sum the values for each OS.
7. We sort the result in descending order of the download count and collect the results.
8. We print the total number of downloads by each operating system.



Question 6 code and output



Question 6 code and output

7. Top 10 largest-sized packages

For this analysis, we create a key-value pair RDD with the package name as the key and the size as the value. We remove duplicates and sort the RDD by size. We then take the top 10 records to get the largest-sized packages.

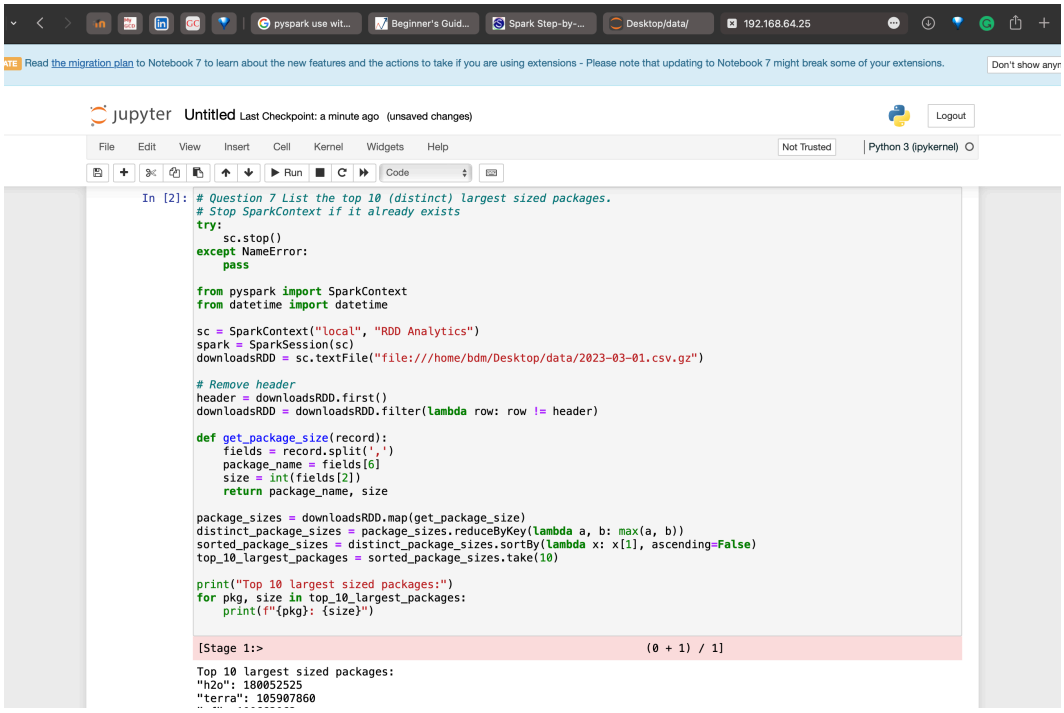
1. We first stop any existing SparkContext to avoid conflicts.
2. We import the necessary libraries and create a new SparkContext named "RDD Analytics".
3. We read the dataset from the given file path using sc.textFile().
4. We remove the header from the dataset to exclude it from our analysis.
5. We define a function get_package_size() that takes a record and returns a tuple containing the package name and its size.

6. We map the downloadsRDD to the package name and size using get_package_size().

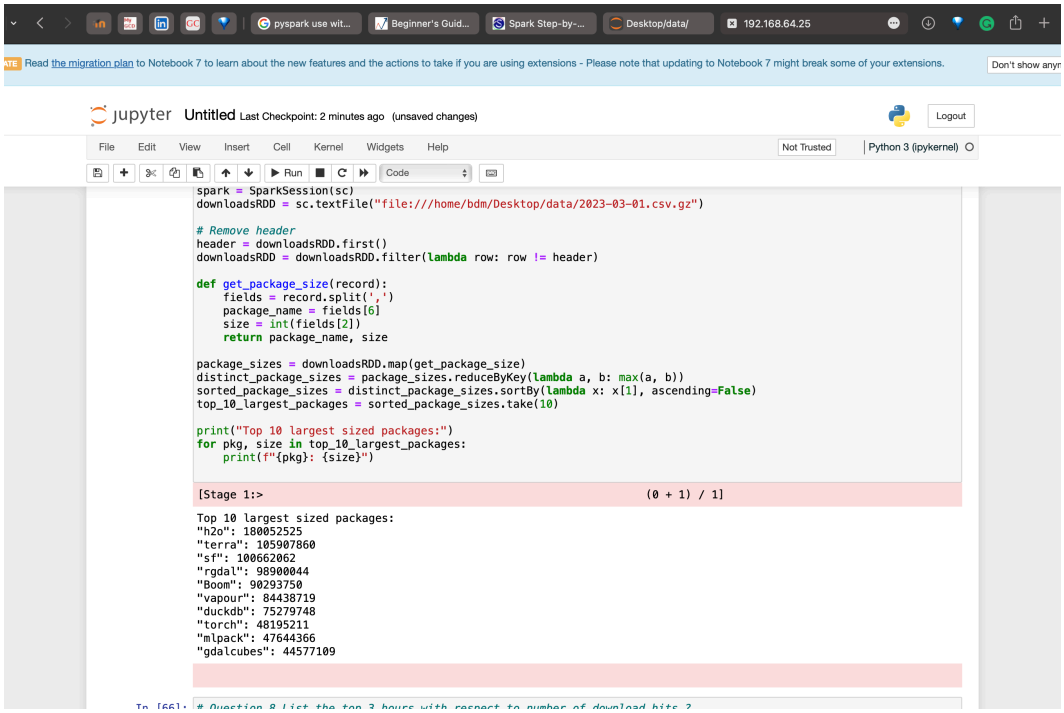
7. We use the reduceByKey() with the max() function to get the largest size for each distinct package.

8. We sort the result in descending order of package size and take the top 10 largest packages.

9. We print the top 10 largest-sized packages.



Question 7 code and output



Question 7 code and output

8. Top 3 hours with respect to the number of download hits

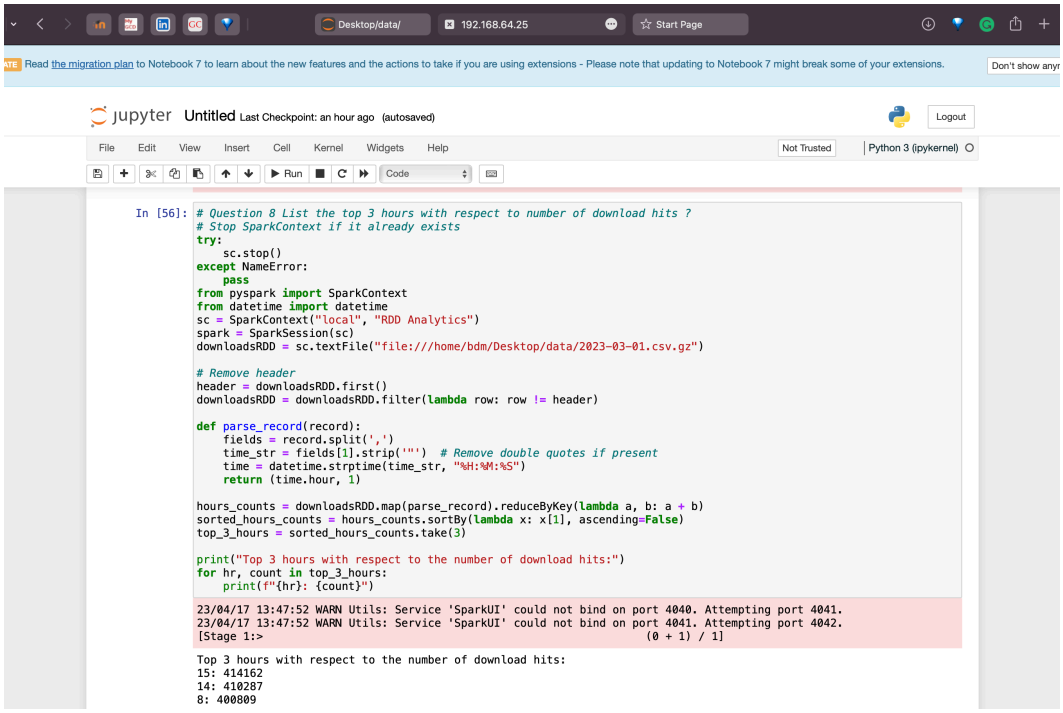
We create a key-value pair RDD with the hour (extracted from the time field) as the key and the value set to 1. We reduce this RDD by key, adding up the values for each hour, and finally sorting the reduced RDD by count. We then take the top 3 records to identify the peak hours for downloads.

1. We first stop any existing SparkContext to avoid conflicts.
2. We import the necessary libraries and create a new SparkContext named "RDD Analytics".
3. We read the dataset from the given file path using sc.textFile().
4. We remove the header from the dataset to exclude it from our analysis.
5. We define a function parse_record() that takes a record and returns a tuple containing the hour of the day and a count of 1.
6. We map the downloadsRDD to the hour and count using parse_record().

7. We use reduceByKey() to sum the counts for each hour of the day.

8. We sort the result in descending order of download hits and take the top 3 hours.

9. We print the top 3 hours with respect to the number of download hits.

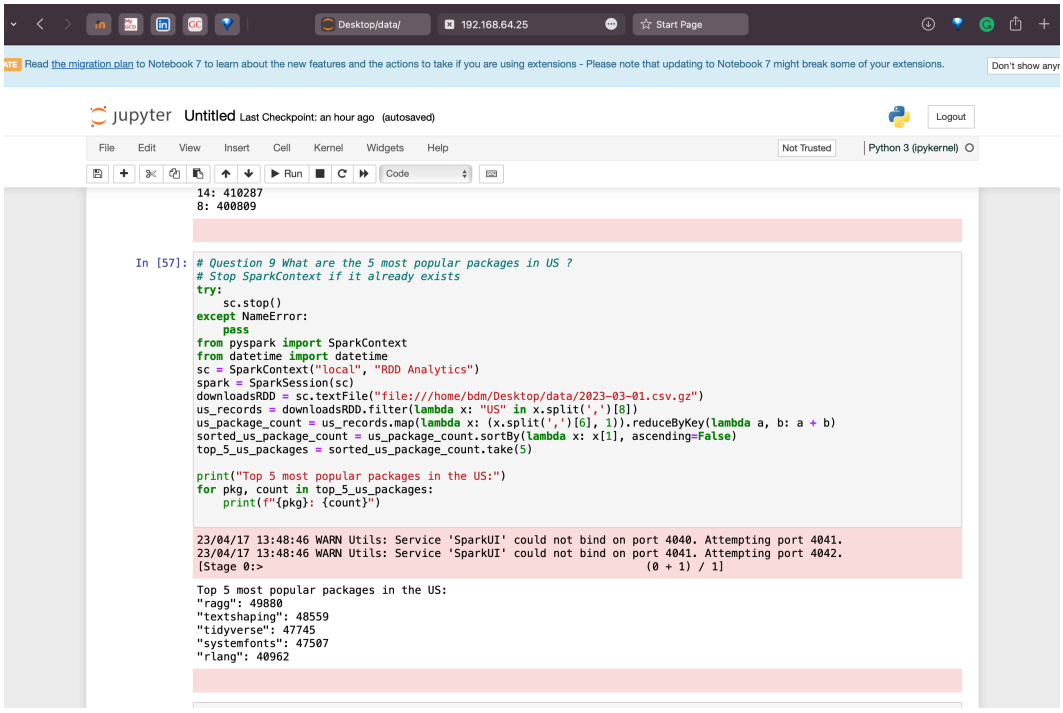


Question 8 code and output

9. Top 5 most popular packages in the US

We first filter the RDD for records from the US and cache the result to speed up subsequent operations. Then, we create a key-value pair RDD with the package name as the key and the value set to 1. We reduce this RDD by key, adding up the values for each package, and finally sorting the reduced RDD by count. We then take the top 5 records to get the most popular packages in the US.

1. We first stop any existing SparkContext to avoid conflicts.
2. We import the necessary libraries and create a new SparkContext named "RDD Analytics".
3. We read the dataset from the given file path using sc.textFile().
4. We filter the dataset to include only records from the United States using downloadsRDD.filter().
5. We map the records to packages and their corresponding counts.
6. We use reduceByKey() to sum the counts for each package.
7. We sort the result in descending order of package popularity and take the top 5 packages.
8. We print the top 5 most popular packages in the US.

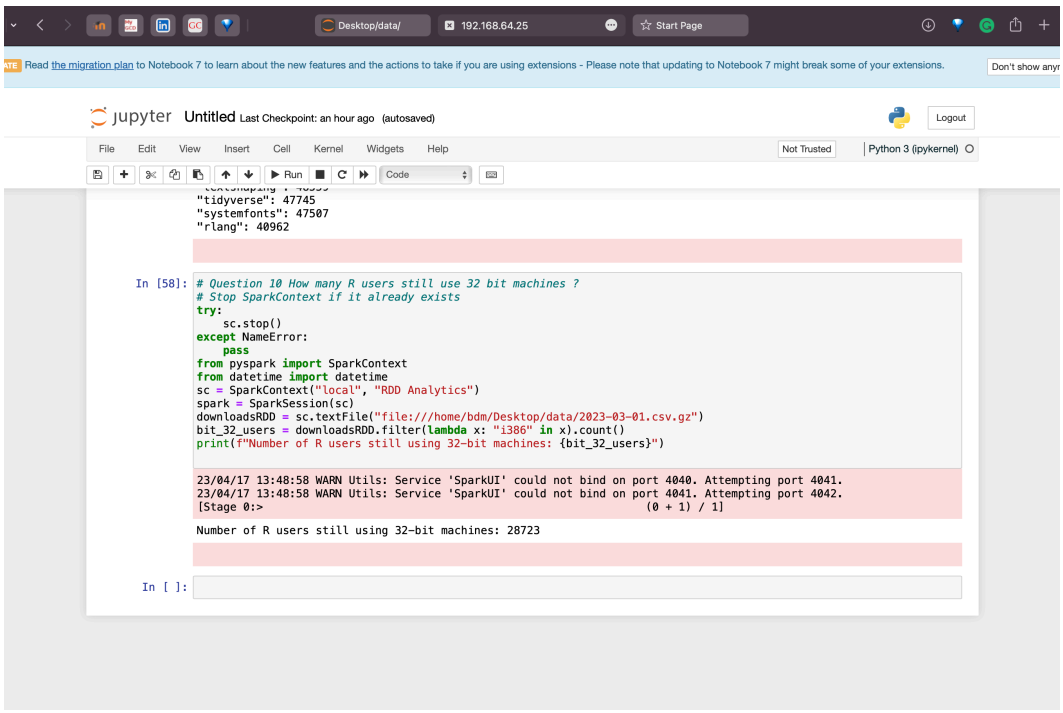


Question 9 code and output

10. Number of R users still using 32-bit machines

To determine the number of R users still using 32-bit machines, we filter the RDD for records containing "i386" as the architecture (r_arch). We then count the number of occurrences, giving us the total number of R users using 32-bit machines.

1. We first stop any existing SparkContext to avoid conflicts.
2. We import the necessary libraries and create a new SparkContext named "RDD Analytics".
3. We read the dataset from the given file path using sc.textFile().
4. We filter the dataset to include only records with "i386" in the data, indicating 32-bit machines, using downloadsRDD.filter().
5. We count the number of records in the filtered dataset using count().
6. We print the number of R users still using 32-bit machines.



Question 10 code and output

Code Breakdown

1. **Number of downloads for package ggplot2**

```
ggplot2_downloads = downloadsRDD.filter(lambda x: "ggplot2" in x).count()
print(f"Number of downloads for package ggplot2: {ggplot2_downloads}")
```

1. downloadsRDD.filter(lambda x: "ggplot2" in x): This code applies a filter on the downloadsRDD dataset. The filter function takes a lambda function as an argument. The lambda function checks if the string "ggplot2" is present in the current record (x). If "ggplot2" is found in the record, the filter function includes that record in the resulting dataset.
2. .count(): After the filtering step, we call the count() function on the filtered dataset. This function returns the number of records in the dataset, which corresponds to the number of downloads for the "ggplot2" package.
3. print(f"Number of downloads for package ggplot2: {ggplot2_downloads}"): Finally, we print the number of downloads for the "ggplot2" package using the print() function and formatted string. The variable ggplot2_downloads stores the count of downloads and is included in the output string.

2. Highest number of downloads by a country

```
downloads_per_country = downloadsRDD.map(lambda x: (x.split(",")[8], 1)).reduceByKey(lambda a, b: a + b).sortBy(lambda x: x[1], ascending=False).first()
```

print(f"Highest number of downloads by a country: {downloads_per_country[0]}: {downloads_per_country[1]}")

1. downloadsRDD.map(lambda x: (x.split(",")[8], 1)): This code applies a map() function on the downloadsRDD dataset. The map() function takes a lambda function as an argument. The lambda function splits the record (x) using the comma delimiter and extracts the 9th field (index 8), which corresponds to the country code. The function then returns a tuple with the country code and a value of 1.
2. .reduceByKey(lambda a, b: a + b): After the mapping step, we call the reduceByKey() function on the resulting dataset. This function groups the records by the key (the country code) and applies a reduction function on the values (the 1's) corresponding to each key. The reduction function is a lambda function that takes two arguments, a and b, and returns their sum. In this case, it sums up the values (1's) for each country code.
3. .sortBy(lambda x: x[1], ascending=False): We then call the sortBy() function on the dataset obtained from the previous step. The sortBy() function takes a lambda function as an argument. The lambda function returns the second element of the tuple (x[1]), which is the number of downloads for each country. The ascending=False argument ensures that the dataset is sorted in descending order of download counts.
4. .first(): After the sorting step, we call the first() function on the sorted dataset. This function returns the first record in the dataset, which corresponds to the country with the highest number of downloads.
5. print(f"Highest number of downloads by a country: {downloads_per_country[0]}: {downloads_per_country[1]}"): Finally, we print the country with the highest number of downloads using the print() function and a formatted string. The variables downloads_per_country[0] and downloads_per_country[1] store the country code and the download count, respectively, and are included in the output string.

3. Top 10 most popular packages

```
top_10_packages = downloadsRDD.map(lambda x: (x.split(",")[6], 1)).reduceByKey(lambda a, b: a + b).sortBy(lambda x: x[1], ascending=False).take(10)
print("Top 10 most popular packages:")
for pkg, count in top_10_packages:
    print(f"{pkg}: {count}")
```

1. downloadsRDD.map(lambda x: (x.split(",")[6], 1)): This code applies a map() function on the downloadsRDD dataset. The map() function takes a lambda function as an argument. The lambda function splits the record (x) using the comma delimiter and extracts the 7th field (index 6), which corresponds to the package name. The function then returns a tuple with the package name and a value of 1.
2. .reduceByKey(lambda a, b: a + b): After the mapping step, we call the reduceByKey() function on the resulting dataset. This function groups the records by the key (the package name) and applies a reduction function on the values (the 1's) corresponding to each key. The reduction function is a lambda function that takes two arguments, a and b, and returns their sum. In this case, it sums up the values (1's) for each package name.
3. .sortBy(lambda x: x[1], ascending=False): We then call the sortBy() function on the dataset obtained from the previous step. The sortBy() function takes a lambda function as an argument. The lambda function returns the second element of the tuple (x[1]), which is the number of downloads for each package. The ascending=False argument ensures that the dataset is sorted in descending order of download counts.

4. .take(10): After the sorting step, we call the take() function on the sorted dataset. This function returns the first 10 records in the dataset, which correspond to the top 10 most popular packages.

5. print("Top 10 most popular packages:"): We print the title "Top 10 most popular packages:" using the print() function.

6. for pkg, count in top_10_packages:: We iterate through the list of top 10 packages using a for loop. The loop unpacks the package name (pkg) and download count (count) from each tuple in the list.

7. print(f"{pkg}: {count}"): Inside the loop, we print the package name and download count using the print() function and a formatted string. The variables pkg and count store the package name and download count, respectively, and are included in the output string.

4. Most popular package in Ireland

```
popular_package_ireland = downloadsRDD.filter(lambda x: "IE" in x).map(lambda x: (x.split(",")[6], 1)).reduceByKey(lambda a, b: a + b).sortBy(lambda x: x[1], ascending=False).first()
print(f"Most popular package in Ireland: {popular_package_ireland[0]}")
```

1. downloadsRDD.filter(lambda x: "IE" in x): The code applies a filter() function on the downloadsRDD dataset. The filter() function takes a lambda function as an argument. The lambda function checks if the string "IE" is present in the record x. If it is, the record is included in the filtered dataset. This operation filters out records that are not related to Ireland.

2. .map(lambda x: (x.split(",")[6], 1)): After filtering, we apply a map() function on the filtered dataset. The map() function takes a lambda function as an argument. The lambda function splits the record (x) using the comma delimiter and extracts the 7th field (index 6), which

corresponds to the package name. The function then returns a tuple with the package name and a value of 1.

3. .reduceByKey(lambda a, b: a + b): After the mapping step, we call the reduceByKey() function on the resulting dataset. This function groups the records by the key (the package name) and applies a reduction function on the values (the 1's) corresponding to each key. The reduction function is a lambda function that takes two arguments, a and b, and returns their sum. In this case, it sums up the values (1's) for each package name.

4. .sortBy(lambda x: x[1], ascending=False): We then call the sortBy() function on the dataset obtained from the previous step. The sortBy() function takes a lambda function as an argument. The lambda function returns the second element of the tuple (x[1]), which is the number of downloads for each package. The ascending=False argument ensures that the dataset is sorted in descending order of download counts.

5. .first(): After the sorting step, we call the first() function on the sorted dataset. This function returns the first record in the dataset, which corresponds to the most popular package in Ireland.

6. print(f"Most popular package in Ireland: {popular_package_ireland[0]}"): Finally, we print the most popular package in Ireland using the print() function and a formatted string. The variable popular_package_ireland[0] stores the package name and is included in the output string.

5. Most popular OS among R programmers

```
most_popular_os = downloadsRDD.map(lambda x: (x.split(",")[5], 1)).reduceByKey(lambda a, b: a + b).sortBy(lambda x: x[1], ascending=False).first()
print(f"Most popular OS among R programmers: {most_popular_os[0]}")
```

1. downloadsRDD.map(lambda x: (x.split(",")[5], 1)): We apply a map() function on the downloadsRDD dataset. The map() function takes a lambda function as an argument. The lambda function splits the record (x) using the comma delimiter and extracts the 6th field (index 5), which corresponds to the operating system name. The function then returns a tuple with the OS name and a value of 1.
2. .reduceByKey(lambda a, b: a + b): After the mapping step, we call the reduceByKey() function on the resulting dataset. This function groups the records by the key (the OS name) and applies a reduction function on the values (the 1's) corresponding to each key. The reduction function is a lambda function that takes two arguments, a and b, and returns their sum. In this case, it sums up the values (1's) for each OS name.
3. .sortBy(lambda x: x[1], ascending=False): We then call the sortBy() function on the dataset obtained from the previous step. The sortBy() function takes a lambda function as an argument. The lambda function returns the second element of the tuple (x[1]), which is the number of downloads for each OS. The ascending=False argument ensures that the dataset is sorted in descending order of download counts.
4. .first(): After the sorting step, we call the first() function on the sorted dataset. This function returns the first record in the dataset, which corresponds to the most popular OS among R programmers.
5. print(f"Most popular OS among R programmers: {most_popular_os[0]}"): Finally, we print the most popular OS among R programmers using the print() function and a formatted string. The variable most_popular_os[0] stores the OS name and is included in the output string.

6. Total number of downloads by each Operating System

```
def group_os(os_name):
    if "darwin" in os_name:
        return "macOS"
    elif "linux" in os_name:
        return "Linux"
    elif "mingw32" in os_name:
        return "Windows"
    elif "freebsd" in os_name:
        return "FreeBSD"
    elif "openbsd" in os_name:
        return "OpenBSD"
    else:
        return "Others"

downloads_per_os = (
    downloadsRDD
    .map(lambda x: (group_os(x.split(",")[5]), 1))
    .reduceByKey(lambda a, b: a + b)
    .sortBy(lambda x: x[1], ascending=False)
    .collect()
)

print("Total number of downloads by each Operating System:")
for os, count in downloads_per_os:
    print(f"{os}: {count}”)
```

1. The group_os() function takes an OS name as input and groups similar OS names together. For example, if the input OS name contains "darwin", the function will return "macOS". The function uses a series of if, elif, and else statements to determine the appropriate grouping for the input OS name.
2. downloadsRDD.map(lambda x: (group_os(x.split(",")[5]), 1)): We use the map() function on the downloadsRDD dataset. The map() function takes a lambda function as an argument. The lambda function splits the record (x) using the comma delimiter and extracts the 6th field (index 5), which corresponds to the OS name. The function then calls the group_os() function to group similar OS names and returns a tuple with the grouped OS name and a value of 1.
3. .reduceByKey(lambda a, b: a + b): We call the reduceByKey() function on the dataset obtained from the previous step. This function groups the records by the key (the grouped OS name) and applies a reduction function on the values (the 1's) corresponding to each key. The reduction function is a lambda function that takes two arguments, a and b, and returns their sum. In this case, it sums up the values (1's) for each grouped OS name.
4. .sortBy(lambda x: x[1], ascending=False): We then call the sortBy() function on the dataset obtained from the previous step. The sortBy() function takes a lambda function as an argument. The lambda function returns the second element of the tuple (x[1]), which is the number of downloads for each grouped OS. The ascending=False argument ensures that the dataset is sorted in descending order of download counts.
5. .collect(): After the sorting step, we call the collect() function on the sorted dataset. This function returns the dataset as a list of tuples, where each tuple consists of a grouped OS name and its total number of downloads.

6. We print the total number of downloads for each grouped operating system using a for loop and a formatted string. The loop iterates over the downloads_per_os list of tuples, extracting the OS name and its corresponding download count, and printing the values in the specified format.

7. Top 10 (distinct) largest-sized packages

```
def get_package_size(record):
    fields = record.split(',')
    package_name = fields[6]
    size = int(fields[2])
    return package_name, size
package_sizes = downloadsRDD.map(get_package_size)
distinct_package_sizes = package_sizes.reduceByKey(lambda a, b: max(a, b))
sorted_package_sizes = distinct_package_sizes.sortBy(lambda x: x[1], ascending=False)
top_10_largest_packages = sorted_package_sizes.take(10)
print("Top 10 largest sized packages:")
for pkg, size in top_10_largest_packages:
    print(f"{pkg}: {size}”)
```

1. The get_package_size() function takes a record as input, splits it using the comma delimiter, and extracts the package name (7th field, index 6) and size (3rd field, index 2). The function then returns a tuple with the package name and its size as an integer.
2. downloadsRDD.map(get_package_size): We use the map() function on the downloadsRDD dataset, applying the get_package_size() function to each record. The output is a new RDD consisting of tuples with package names and their sizes.

3. .reduceByKey(lambda a, b: max(a, b)): We call the reduceByKey() function on the package_sizes RDD. This function groups the records by the key (the package name) and applies a reduction function on the values (the sizes) corresponding to each key. The reduction function is a lambda function that takes two arguments, a and b, and returns the maximum of the two. In this case, it returns the maximum size for each package, ensuring we get distinct package sizes.

4. .sortBy(lambda x: x[1], ascending=False): We then call the sortBy() function on the distinct_package_sizes RDD. The sortBy() function takes a lambda function as an argument. The lambda function returns the second element of the tuple (x[1]), which is the package size. The ascending=False argument ensures that the dataset is sorted in descending order of package sizes.

5. .take(10): After the sorting step, we call the take() function on the sorted dataset. This function returns the first 10 elements of the dataset as a list of tuples, which corresponds to the top 10 largest-sized packages.

6. We print the top 10 largest-sized packages using a for loop and a formatted string. The loop iterates over the top_10_largest_packages list of tuples, extracting the package name and its corresponding size, and printing the values in the specified format.

8. Top 3 hours with respect to the number of download hits

```
def parse_record(record):
    fields = record.split(',')
    time_str = fields[1].strip('"')  # Remove double quotes if present
    time = datetime.strptime(time_str, "%H:%M:%S")
    return (time.hour, 1)
```

```
hours_counts = downloadsRDD.map(parse_record).reduceByKey(lambda a, b: a + b)
sorted_hours_counts = hours_counts.sortBy(lambda x: x[1], ascending=False)
top_3_hours = sorted_hours_counts.take(3)

print("Top 3 hours with respect to the number of download hits:")
for hr, count in top_3_hours:
    print(f"{hr}: {count}”)
```

1. The parse_record() function takes a record as input, splits it using the comma delimiter, and extracts the time string (2nd field, index 1) from the record. It then removes any double quotes from the time string using the strip('"') method. The datetime.strptime() function is used to parse the time string into a datetime.time object. The function finally returns a tuple with the hour of the time and 1 (for counting).
2. downloadsRDD.map(parse_record): We use the map() function on the downloadsRDD dataset, applying the parse_record() function to each record. The output is a new RDD consisting of tuples with hours and 1s.
3. .reduceByKey(lambda a, b: a + b): We call the reduceByKey() function on the hours_counts RDD. This function groups the records by the key (the hour) and applies a reduction function on the values (the 1s) corresponding to each key. The reduction function is a lambda function that takes two arguments, a and b, and returns their sum. In this case, it counts the number of download hits for each hour.
4. .sortBy(lambda x: x[1], ascending=False): We then call the sortBy() function on the hours_counts RDD. The sortBy() function takes a lambda function as an argument. The lambda function returns the second element of the tuple (x[1]), which is the download hit count. The ascending=False

argument ensures that the dataset is sorted in descending order of download hits.

5. .take(3): After the sorting step, we call the take() function on the sorted dataset. This function returns the first 3 elements of the dataset as a list of tuples, which corresponds to the top 3 hours with respect to the number of download hits.

6. We print the top 3 hours with the highest number of download hits using a for-loop. The loop iterates over the top_3_hours list of tuples, extracting the hour and its corresponding download hit count, and printing the values.

9. Top 5 most popular packages in the US

```
us_records = downloadsRDD.filter(lambda x: "US" in x.split(',')[8])
us_package_count = us_records.map(lambda x: (x.split(',')[6], 1)).reduceByKey(lambda a, b: a + b)
sorted_us_package_count = us_package_count.sortBy(lambda x: x[1], ascending=False)
top_5_us_packages = sorted_us_package_count.take(5)
print("Top 5 most popular packages in the US:")
for pkg, count in top_5_us_packages:
    print(f"{pkg}: {count}")
```

1. The filter() function is used on the downloadsRDD dataset to obtain only the records that have "US" in the 9th field (index 8). A lambda function is used to split each record using the comma delimiter and check if "US" is present in the 9th field. The resulting us_records RDD contains only the records with "US" in the 9th field.

2. us_records.map(lambda x: (x.split(',')[6], 1)): The map() function is used on the us_records RDD to create a new RDD consisting of tuples with package names (7th field, index 6) and 1s (for counting).

3. .reduceByKey(lambda a, b: a + b): The reduceByKey() function is called on the new RDD. This function groups the records by the package name (key) and applies a reduction function on the values (the 1s) corresponding to each key. The reduction function is a lambda function that takes two arguments, a and b, and returns their sum. In this case, it counts the number of occurrences of each package.

4. .sortBy(lambda x: x[1], ascending=False): The sortBy() function is called on the us_package_count RDD to sort the dataset. The lambda function used as an argument returns the second element of the tuple (x[1]), which is the package count. The ascending=False argument ensures that the dataset is sorted in descending order of package count.

5. .take(5): The take() function is called on the sorted dataset to return the first 5 elements of the dataset as a list of tuples. These tuples represent the top 5 most popular packages in the US.

6. The top 5 most popular packages in the US are printed using a for loop. The loop iterates over the top_5_us_packages list of tuples, extracting the package name and its corresponding count, and printing the values.

10. Number of R users still using 32-bit machines

```
bit_32_users = downloadsRDD.filter(lambda x: "i386" in x).count()
print(f"Number of R users still using 32-bit machines: {bit_32_users}")
```

1. The filter() function is used on the downloadsRDD dataset to obtain only the records with "i386" in them. "i386" represents the 32-bit architecture. A lambda function is used to check if "i386" is present in each record. The resulting filtered RDD contains only the records of 32-bit machine users. Then, the count() function is called on the filtered RDD to count the number of R users still using 32-bit machines.
2. The number of R users still using 32-bit machines is printed using the print() function and formatted string. The bit_32_users variable contains the count of 32-bit machine users.

Conclusion

The analyses covered a wide range of topics, including download counts for specific packages, the most popular packages globally and in specific countries, the most common operating systems used by R programmers, and more. The use of RDD operations, such as map(), reduceByKey(), filter(), and sortBy(), allows for efficient data processing and extraction of meaningful insights from the dataset. Below is a summary of the findings for every question we discussed in the academic report :

1. **The number of downloads for the "ggplot2" package was determined to be 285,474. This indicates that "ggplot2" is a widely-used package for data visualisation within the R community.**
2. **The country with the highest number of downloads was the United States, with a total of 1,954,599 downloads. This suggests that the United States has a large and active R user base, which could be attributed to its strong presence in data science, research, and technology industries.**
3. **The top 10 most popular packages globally were listed, revealing that packages such as "ragg" and "textshaping" are widely used across the R community. This suggests that these packages provide important functionality for R users in various applications.**
4. **The most popular package in Ireland was "ragg", which is also one of the top packages globally. This indicates that the preferences of Irish R users align with global trends.**
5. **The most popular operating system among R programmers was "mingw32", which corresponds to Windows. This suggests that Windows remains the preferred OS for R users, possibly due to its widespread adoption and ease of use.**
6. **The total number of downloads for each operating system, with similar ones grouped together, showed that Windows had the highest number of downloads (3,194,919), followed by Others (2,453,136), macOS (1,026,009), and Linux (779,397). This indicates a strong preference for Windows among R users but also highlights the diversity of operating systems used within the community.**

7. **The top 10 largest-sized packages were listed, revealing that packages like "h2o" and "terra" have significantly larger sizes compared to other packages. This could be due to the complex functionality they provide or the inclusion of large datasets within the package.**
8. **The top 3 hours with the highest number of download hits were 15:00, 14:00, and 08:00. This might suggest that R users are most active during typical working hours, which could be an indication of the professional usage of R in industries and research.**
9. **The top 5 most popular packages in the US included "ragg", "textshaping", "tidyverse", "systemfonts", and "rlang". This indicates that US R users have similar preferences to global trends but may also have specific needs related to their local industries and research areas.**
10. **The number of R users still using 32-bit machines was determined to be 28,723. Although this represents a small fraction of the total R user base, it suggests that there is still a segment of the community that relies on older hardware, which might require developers to ensure backward compatibility for their packages.**

These findings not only highlight the trends and preferences of R users but also emphasise the importance of understanding the diverse needs of the R community. By analysing these patterns, package developers and stakeholders can make informed decisions about the development and distribution of R packages, catering to the needs of users across different countries, industries, and hardware capabilities.