

به نام خدا

تمرین سوم
Spark

امیرمحمد رنجبر پازکی ۸۱۰۱۹۹۳۴۰

درس تحلیل داده‌های حجیم

دانشکده‌ی مهندسی برق و کامپیوتر
بهار ۱۴۰۰

• گام اول - دستورات پایه:

در ابتدا نیاز است تا تنظیمات موردنظر بر روی colab انجام شود تا بتوان با spark کار کرد.
برای این کار ابتدا باید جاوا و اسپارک نصب شود.

Install requirements

```
[ ] !apt-get install openjdk-8-jdk-headless -qq > /dev/null
!wget -q http://archive.apache.org/dist/spark/spark-3.1.1/spark-3.1.1-bin-hadoop3.2.tgz
!tar xf spark-3.1.1-bin-hadoop3.2.tgz
!pip install -q findspark
```

سپس، مسیرهای SPARK_HOME و JAVA_HOME باید تعریف شوند تا بتوان از دستورات آنها استفاده کرد.

Set environment variables

```
[ ] import os
os.environ[ "JAVA_HOME" ] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ[ "SPARK_HOME" ] = "/content/spark-3.1.1-bin-hadoop3.2"
```

حال spark و دیگر کتابخانه‌های موردنیاز import می‌شود و با تنظیماتی از spark یک session برای کار با آن گرفته می‌شود.

Import libraries

```
▶ import findspark
findspark.init()
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[*]").getOrCreate()
spark.conf.set("spark.sql.repl.eagerEval.enabled", True) # Property used to format output tables better
spark
```

SparkSession - in-memory

SparkContext

Spark UI

Version

v3.1.1

Master

local[*]

AppName

pyspark-shell

```
[ ] import re
```

سپس، sparkContext را می‌گیریم تا بتوان با استفاده از آن داده‌ها را خواند.

Get spark context

```
[ ] sc = spark.sparkContext
```

بعد با استفاده از `textFile` فایل ورودی این سوال خوانده می‌شود.

Read input file

```
▶ input_file_path = "/content/input.txt"
input_file = sc.textFile(input_file_path)
print(input_file.first())
```

⇒ Games are a fun way to get people involved and learning in a happy environment and get them to work on concepts

+ بخش اول:

در این بخش با استفاده از دستور `flatMap` همه خطوط با استفاده از فاصله شکسته‌می‌شوند و کلمات حاصل در یک آرایه خروجی قرار می‌گیرد. سپس، با استفاده از دستور `count` تعداد لغات این فایل شمرده شده‌است. خروجی در زیر قابل مشاهده‌است. تعداد کلمات ۵۰۴۳ بوده‌است.

Count number of words

```
[ ] wordsRDD = input_file.flatMap(lambda line: line.split(" "))
print("First 5 words:")
print(wordsRDD.take(5))
words_count = wordsRDD.count()
print("Words count: {}".format(words_count))
```

```
First 5 words:
['Games', 'are', 'a', 'fun', 'way']
Words count: 5043
```

سپس، تعداد تکرار هر کلمه محاسبه شده‌است و ضمن نمایش در فایل `words_count.txt` ذخیره شده‌است و در کنار کتابچه این گام بارگذاری شده‌است.

Count each word repeat and save output in words_count.txt

```
[ ] print("Words repeat count:")
words_repetition = wordsRDD.countByValue()
print(words_repetition)
output_file_name = "words_count.txt"
with open(output_file_name, "w") as output_file:
    for word, count in words_repetition.items():
        new_line = f"{word}: {count}\n"
        output_file.write(new_line)
output_file.close()
```

Words repeat count:
defaultdict(<class 'int'>, {'Games': 1, 'are': 31, 'a': 106, 'fun': 2, 'way': 16, 'to': 151, 'get': 18, 'people': 51,
حال علائم نقطه‌گذاری حذف شده است و مراحل بالا دوباره تکرار شده است. برای این کار، در جدا
سازی کلمات یک خط از علائم نقطه‌گذاری نیز استفاده شده است و همچنین با یک فیلتر چک
شده که این کلمات پوج نباشند. در این قسمت ضمن نمایش خروجی تمیزشده در فایل
شده که این کلمات ذخیره شده است و در کنار کتابچه این گام بارگذاری شده است. تعداد کلمات
words_count.txt پس از حذف علائم ۵۰۷ بوده است. تعداد تکرار کلمات نیز در انتهای قابل مشاهده است.

Remove punctuation marks and repeat actions

```
[ ] clean_wordsRDD = (input_file
    .flatMap(lambda line: re.split('[ ?!.,;:\t()]', line))
    .filter(lambda word: word != ''))
print("First 5 words:")
print(clean_wordsRDD.take(5))
clean_words_count = clean_wordsRDD.count()
print("Words count: {}".format(clean_words_count))

First 5 words:
['Games', 'are', 'a', 'fun', 'way']
Words count: 5007
```

```
▶ print("Words repeat count:")
clean_words_repetition = clean_wordsRDD.countByValue()
print(clean_words_repetition)
output_file_name = "clean_words_count.txt"
with open(output_file_name, "w") as output_file:
    for word, count in clean_words_repetition.items():
        new_line = f"{word}: {count}\n"
        output_file.write(new_line)
output_file.close()

⇒ Words repeat count:
defaultdict(<class 'int'>, {'Games': 1, 'are': 32, 'a': 106, 'fun': 2, 'way': 17, 'to': 152, 'get': 18, 'people': 57,
```

+ بخش دوم:

تمامی کلماتی که با m/M شروع می‌شوند با استفاده از یک فیلتر استخراج شده‌اند و تعداد آن‌ها با استفاده از count شمرده شده است. خروجی در زیر قابل مشاهده است. تعداد این کلمات ۱۵۰ است.

Find words count that start with "m/M"

```
▶ words_with_m = clean_wordsRDD.filter(lambda word: word[0] in ["m", "M"])
print("First 5 words:")
print(words_with_m.take(5))
words_with_m_count = words_with_m.count()
print("Words starts with m count: {}".format(words_with_m_count))
```

```
⇨ First 5 words:
['make', 'make', 'mastery', 'many', 'me']
Words starts with m count: 150
```

+ بخش سوم:

در این بخش ابتدا با استفاده از یک فیلتر و چک کردن طول کلمات لغات ۵ حرفی موجود جدا شده‌اند. تعداد این لغات ۵۸۱ بوده‌است که ۵ عدد از آن‌ها در خروجی زیر قابل مشاهده‌است.

Count words with 5 chars

```
▶ five_char_words = clean_wordsRDD.filter(lambda word: len(word) == 5)
print("First 5 words:")
print(five_char_words.take(5))
five_char_words_count = five_char_words.count()
print("Five char words: {}".format(five_char_words_count))
```

```
⇨ First 5 words:
['Games', 'happy', 'these', 'games', 'class']
Five char words: 581
```

سپس، حروفی که در ابتدایشان حرف صدادار بوده‌است با استفاده فیلتر دیگری کنار گذاشته شده‌اند و با استفاده از `takeOrdered` خروجی همه کلمات به صورت مرتب شده نمایش داده شده‌است.
تعداد کلمات ۵ حرفی که با حروف صدا دار شروع نمی‌شوند، ۵۰۷ بوده‌است. خروجی مرتب شده در زیر قابل مشاهده‌است.

Remove words starts with vowels

```
[12] vowels = ["a", "u", "i", "o", "u"]
reduced_five_words = five_char_words.filter(lambda word: word.lower()[0] not in vowels)
print("First 5 words:")
print(reduced_five_words.take(5))
reduced_five_words_count = reduced_five_words.count()
print("Five char words without vowel start: {}".format(reduced_five_words_count))
print("Sorted result:")
print(reduced_five_words.takeOrdered(reduced_five_words_count))
```

```
First 5 words:
['Games', 'happy', 'these', 'games', 'class']
Five char words without vowel start: 507
Sorted result:
['Buddy', 'Buddy', 'Buddy', 'Buddy', 'Buddy', 'Court', 'Cross', 'Final', 'Games', 'Given', 'Greek', 'Hooks',
```

+ بخش چهارم:

برای انجام این بخش ابتدا تعداد کلماتی که در بخش اول شمرده شده بود بر مبنای تعداد مرتب شده‌اند. سپس، تعداد ۱۰ درصد محاسبه شده و از این کلمات جدا می‌شود و در لیستی با نام stop_words ذخیره می‌شوند.

Find stop words

```
▶ sorted_repeation = sorted(clean_words_repeation.items(), key=lambda item: item[1], reverse=True)
ten_percent_count = len(sorted_repeation) // 10
stop_words_with_count = sorted_repeation[:ten_percent_count]
stop_words = [stop_word for stop_word, _ in stop_words_with_count]
print("First 10 stop words:")
print(stop_words[:10])
```



```
⇨ First 10 stop words:
['the', 'and', 'of', 'to', 'in', 'a', 'is', 'that', 'people', 'it']
```

سپس، تابعی با عنوان clean_line نوشته شد که با گرفتن یک خط تمام کاراکترهای غیرالفبایی را حذف کرده و ایست واژه‌ها را نیز از بین می‌برد و در نهایت، خط تمیز شده را بر می‌گرداند. سپس، این تابع رو هر خط اجرا می‌شود و نتیجه در فایل removed_stop_words ذخیره می‌شود. این فایل در کنار کتابچه پروژه بارگذاری شده است.

Remove stopwords and non alpha numeric chars

```
!rm -r removed_stop_words
def clean_line(line):
    cleaned_line = ""
    for word in line.split():
        if word not in stop_words:
            cleaned_line += word
            cleaned_line += " "
    cleaned_line = re.sub("[^0-9a-zA-Z]+", " ", cleaned_line)
    return cleaned_line

clean_lines = (input_file.flatMap(lambda line: line.split(".")))
              .map(clean_line)
clean_lines.collect()
removed_stop_words_file_name = "removed_stop_words"
clean_lines.coalesce(1).saveAsTextFile(removed_stop_words_file_name)
```

بخشی از خروجی این فایل به صورت زیر است.

Games fun involved learning happy environment work concepts tactics without knowing lot time Because this perfect negotiation persuasion loosened allowed learn fun environment reinforcing concepts talked got familiar lectures safe give spin test drive persuasion tactics our peers With able connection between theory application concepts no paper learning language framing extent want identify subject level before persuade act concepts theory application takes practice mastery Rufo shown us anything possible five total games Bullshit Car salesman Werewolf XY final goes names For purposes paper referred Fuck Your Buddy another game called Win You Lose since precursor debate off didn't count it several tactics lot overlapped ones I'm going discuss lying identification collaboration three strongest persuasion tactics found playing games often admitting straight away help long run knowing hear teaming up I'm stronger persuading than easier rid competition combination tactics allowed excel played strong grasp theory focused on Lying great Lying instantly hear greater chance trust you first facet I'm going talk about outright lying XY Bullshit outright real exceed game Bullshit don't work telling lie exactly truth don't question saying found best ways start conversation passionate about story nod affirm while slip five sevens down table People talking themselves winning card everybody wins useful XY game so gonna play X point last minute write Y reap rewards Five sevens might seem exaggeration actually away it Lying great Secondly immediately telling lied helped trust expect lie immediately I'm sort person gives expectation honest against later point If i say outlandish right away lie better chance believing me Lying great leads Identification Getting competitors gives huge leg comes Fuck Your Buddy Werewolf game Fuck Your Buddy tactic Identify group lead believe grasp while slowly suggesting moves explaining rules allowed sway results favour What show cards too early won practise rounds ganged graded round kicked out quickly need sure identification others isn't tarnished fact I'm actually talk For reason that's gotten reputation accurate burned Fuck Your Buddy Werewolf game though killed early on classmates act differently wolf trying blend if audible members class speaking if seen quiet classmate made quite easy identify early wolves were acted identify try swaying public opinion kill lead victory townspeople Identification pretty powerful appropriately Collaboration useful games closely tied identification lead teaming early games using defeat better players knowing best comes Collaboration negotiation using people useful tactic Fuck Buddy XY collaboration best both until enough power rid end Fuck Your Buddy beginning just learning collaborating getting faster than others equal grasp rules regulations extremely useful tactic team against players outed quite quickly identified threat three players collaborated rid me After gone smaller teams trying rid other always teams until bitter end XY called collaboration concept greater good extra credit side bonus rounds enough points power can't catch up Collaboration basically getting trust need it Very competitive useful All taught pay attention say talking people especially certain act certain way calss taught focus mastering language persuasion tool rather than merely method conveying information practical applications things learned class now go out world persuade bidding convincing give money just three fifty Part 1 Persepolis fantastic portrayal pain love

بخش پنجم:

برای انجام این بخش ابتدا دو کلمه‌ای‌ها پیدا شده‌اند. برای این کار ابتدا خطوط تفکیک شده‌اند و سپس، کلمات هر خط جدا شده‌اند. سپس، کلمات بقل هم در هر خط با هم tuple شده‌اند. خروجی این بخش به صورت زیر است.

Find bigrams

```
▶ bigrams = input_file.flatMap(lambda line: line.split(".")) \
    .map(lambda line: line.strip().split(" ")) \
    .flatMap(lambda xs: (tuple(x) for x in zip(xs, xs[1:])))

print("First five bigrams:")
print(bigrams.take(5))
```

```
First five bigrams:
[('Games', 'are'), ('are', 'a'), ('a', 'fun'), ('fun', 'way'), ('way', 'to')]
```

در این گام تعداد هر کدام از این دو کلمه‌ای‌ها با استفاده از `countByValue` شمرده شده است و سپس خروجی با استفاده از `sorted` مرتب شده است و نمایش داده شده است. خروجی و کد این بخش به صورت زیر است.

Calculate frequencies and sort them

```
▶ bigrams_frequencies = bigrams.countByValue()
sorted_bigram_frequencies = sorted(bigrams_frequencies.items(), key=lambda item: item[1], reverse=True)
print("Bigrams: (Sorted by frequency)")
for word, repeat in sorted_bigram_frequencies:
    print(f"{word}: {repeat}")
```

```
▶ Bigrams: (Sorted by frequency)
```

```
('of', 'the'): 56
('in', 'the'): 48
('and', 'the'): 15
('all', 'of'): 12
('on', 'the'): 12
('to', 'the'): 11
('people', 'to'): 10
('of', 'a'): 10
('to', 'get'): 9
('in', 'a'): 9
('of', 'this'): 9
('with', 'the'): 9
('it', 'is'): 9
('Soho', 'Square'): 9
('This', 'is'): 8
('the', 'square'): 8
('them', 'to'): 7
('is', 'the'): 7
('this', 'is'): 7
('one', 'of'): 7
('the', 'way'): 7
('for', 'the'): 7
('have', 'been'): 7
('in', 'this'): 6
('will', 'be'): 6
('have', 'a'): 6
('is', 'a'): 6
('that', 'they'): 6
('the', 'game'): 6
('and', 'then'): 6
('it', 'was'): 6
('as', 'a'): 6
```

• گام دوم - بررسی یک فایل لگ وب سرور:

در ابتدا تنظیمات را مشابه گام قبلی انجام می‌دهیم.

تنها تفاوت آن است که چند کتابخانه دیگر نیز import شده‌اند. همچنین داده‌ها به دلیل حجم بودن روی درایو قرار گرفته‌اند و به همین دلیل، نیاز است تا import drive نیز شود تا با آن بتوان درایو را mount کرد و داده‌ها را از آن خواند.

```
[4] import re
    import datetime
    import matplotlib.pyplot as plt
    from google.colab import drive
    from pyspark.sql import Row
```

Mount drive for log file



```
drive.mount('/content/drive')
```

```
↳ Mounted at /content/drive
```

فرمت لگ وب سرور برای کار مناسب نیست چرا که فرمت استاندارد نیست. به همین دلیل، با استفاده از یک regex بخش‌های مختلف را از آن بیرون می‌کشیم. این regex به صورت زیر است.

```
LOG_PATTERN = '^(\s+) (\s+) (\s+) \[(\w:/]+\s+[-]\d{4})\] "(\s+) (\s+) (\s+)" (\d{3}) (\s+)'
```

حال با استفاده از تابع log هر parse_single_log ای که ورودی بگیرد یک row شامل فیلد‌های مختلف موجود خروجی می‌دهد. در این تابع از regex موردنظر و تابع parse_log_time استفاده شده‌است. تابع parse_log_time یک datetime شامل سال و ماه و روز و ساعت و دقیقه و ثانیه می‌سازد و خروجی می‌دهد تا کار با آن راحت‌تر باشد. این تکه‌کدها در زیر قابل مشاهده است.

Extract fields from data function

```
▶ MONTH_NUMBER_MAPPING = {'Jan': 1, 'Feb': 2, 'Mar': 3, 'Apr': 4, 'May': 5, 'Jun': 6, 'Jul': 7, 'Aug': 8, 'Sep': 9, 'Oct': 10, 'Nov': 11, 'Dec': 12}
LOG_PATTERN = '^(\S+) (\S+) (\S+) \[(\w:/]+\s[\+\-]\d{4})\] "(\S+) (\S+) (\S+)" (\d{3}) (\S+)'

def parse_log_time(time):
    return datetime.datetime(int(time[7:11]),
                            MONTH_NUMBER_MAPPING[int(time[3:6])],
                            int(time[0:2]),
                            int(time[12:14]),
                            int(time[15:17]),
                            int(time[18:20]))

def parse_single_log(log):
    match = re.search(LOG_PATTERN, log)
    if match is None:
        return (log, 0)
    size_field = match.group(9)
    if size_field == '-':
        size = 0
    else:
        size = int(match.group(9))
    return (Row(
        host      = match.group(1),
        client_id = match.group(2),
        user_id   = match.group(3),
        date_time = parse_log_time(match.group(4)),
        method    = match.group(5),
        endpoint  = match.group(6),
        protocol  = match.group(7),
        response_code = int(match.group(8)),
        content_size = size
    ), 1)
```

سپس، با استفاده از sparkContext فایل موردنظر خوانده می‌شود و اولین خط آن چاپ می‌شود.

Read web server log file

```
[8] log_file_path = '/content/drive/MyDrive/BD Logs/Log'
pure_logs = sc.textFile(log_file_path)
print("First line of log:")
print(pure_logs.first())

First line of log:
199.72.81.55 - - [01/Jul/1995:00:00:01 -0400] "GET /history/apollo/ HTTP/1.0" 200 6245
```

سپس، استخراج اطلاعات روی هر لگ اجرا می‌شود و در قالب یک ROW ذخیره می‌شود.

تابع استخراج اطلاعات اگر بتواند استخراج موفقیت آمیز داشته باشد ۱ برمی‌گرداند و اگر نتواند صفر برمی‌گرداند. خروجی این استخراج اطلاعات به صورت زیر است.

Extract fields from data

```
[ ] parsed_logs = (pure_logs
    .map(parse_single_log))
print("First parsed log:")
print(parsed_logs.first())
print("Parsed Log Count: {}".format(parsed_logs.count()))

successful_parsed = (parsed_logs
    .filter(lambda record: record[1] == 1)
    .map(lambda record: record[0]))
print("First successfully parsed:")
print(successful_parsed.first())
print("Successful Parse Count: {}".format(successful_parsed.count()))

failed_parsed = (parsed_logs
    .filter(lambda record: record[1] == 0)
    .map(lambda record: record[0]))
print("First fail parsed:")
print(failed_parsed.first())
print("Failed Parse Count: {}".format(failed_parsed.count()))

First parsed log:
(Row(host='199.72.81.55', client_identd='-', user_id='-', date_time=datetime.datetime(1995, 7, 1, 0, 0, 1), method='GET', endpoint='/history/apollo', protocol='HTTP/1.0')
Paredes Log Count: 1891715
First successfully parsed:
Row(host='199.72.81.55', client_identd='-', user_id='-', date_time=datetime.datetime(1995, 7, 1, 0, 0, 1), method='GET', endpoint='/history/apollo', protocol='HTTP/1.0',
Successful Parse Count: 1886838
First fail parsed:
pipe6.nyc.pipeline.com - - [01/Jul/1995:00:22:43 -0400] "GET /shuttle/missions/sts-71/movies/sts-71-mir-dock.mpg" 200 946425
Failed Parse Count: 4877
```

همانطور که مشاهده می‌کنید، تعداد fail شدن‌ها بسیار بالاست. به همین دلیل، الگوی مورد استفاده در regex را بالاستفاده از نمونه fail شده بهتر می‌کنیم. مشکل این است که ممکن است پروتکل در انتهای لایگ نباشد. الگوی ارتقا پیدا کرده و نتیجه‌ی استخراج در زیر قابل مشاهده است.

Improve log pattern

```
LOG_PATTERN = '^(\S+) (\S+) (\S+) \[(\w:/]+\s[\+\-]\d{4}\] "(\S+) (\S+)\s*(\S*)" (\d{3}) (\S+)'
parsed_logs = (pure_logs
    .map(parse_single_log))
print("First parsed log:")
print(parsed_logs.first())
print("Parsed Log Count: {}".format(parsed_logs.count()))

successful_parsed = (parsed_logs
    .filter(lambda record: record[1] == 1)
    .map(lambda record: record[0]))
print("First successfully parsed:")
print(successful_parsed.first())
print("Successful Parse Count: {}".format(successful_parsed.count()))

failed_parsed = (parsed_logs
    .filter(lambda record: record[1] == 0)
    .map(lambda record: record[0]))
print("First fail parsed:")
print(failed_parsed.first())
print("Failed Parse Count: {}".format(failed_parsed.count()))

First parsed log:
(Row(host='199.72.81.55', client_identd='-', user_id='-', date_time=datetime.datetime(1995, 7, 1, 0, 0, 1), method='GET', endpoint='/history/apollo', protocol='HTTP/1.0')
Paredes Log Count: 1891715
First successfully parsed:
Row(host='199.72.81.55', client_identd='-', user_id='-', date_time=datetime.datetime(1995, 7, 1, 0, 0, 1), method='GET', endpoint='/history/apollo', protocol='HTTP/1.0',
Successful Parse Count: 1890851
First fail parsed:
204.120.229.63 - - [01/Jul/1995:04:29:05 -0400] "GET /history/history.html
hqpao/hqpao_home.html HTTP/1.0" 200 1502
Failed Parse Count: 864
```

یک مرحله دیگر بهبود صورت گرفته است که نتیجه آن در زیر قابل مشاهده است.

More improvement for log pattern :D

```
LOG_PATTERN = '^(\S+) (\S+) (\S+) \(((\w:/)+\s[+\-]\d{4})\) "(\S+) (\S+)\s*(\S*)\s*" (\d{3}) (\S+)'
parsed_logs = (pure_logs
    .map(parse_single_log))
print("First parsed log:")
print(parsed_logs.first())
print("Paresed Logs Count: {}".format(parsed_logs.count()))

successful_parsed = (parsed_logs
    .filter(lambda record: record[1] == 1)
    .map(lambda record: record[0]))
    .cache()
print("First successfully parsed:")
print(successful_parsed.first())
print("Successful Parse Count: {}".format(successful_parsed.count()))

failed_parsed = (parsed_logs
    .filter(lambda record: record[1] == 0)
    .map(lambda record: record[0]))
    .cache()
print("First 10 fail parsed:")
print(failed_parsed.take(10))
print("Failed Parse Count: {}".format(failed_parsed.count()))

First parsed log:
(Row(host='199.72.81.55', client_identd='-', user_id='-', date_time=datetime.datetime(1995, 7, 1, 0, 0, 1), method='GET', endpoint='/history/apollo', protocol='HTTP/1.1')
Parsed Logs Count: 1891715
First successfully parsed:
Row(host='199.72.81.55', client_identd='-', user_id='-', date_time=datetime.datetime(1995, 7, 1, 0, 0, 1), method='GET', endpoint='/history/apollo', protocol='HTTP/1.1')
Successful Parse Count: 1890866
First 10 fail parsed:
['204.120.229.63 - - [01/Jul/1995:04:29:05 -0400] "GET /history/history.html
Failed Parse Count: 849
hqpaohqpaohome.html HTTP/1.0" 200 1502',
```

تعداد fail شده‌ها قابل قبول است و نمونه‌ای که وجود دارد دو url دارد که مطلوب نیست و مشکل دار است.
+ بخش اول:

برای شمردن تعداد host‌های یکتا در این فایل کافیست host‌ها را جدا کرده و روی آنها distinct اجرا کرده تا تکراری‌ها حذف شوند.
حال با استفاده از دستور count می‌توان آنها را شمرد. خروجی در زیر قابل مشاهده است. در این فایل ۸۱۹۷۱ host یکتا وجود دارد.

Part 1

```
unique_hosts = (successful_parsed
    .map(lambda log: log.host)
    .distinct())
print("First 5 hosts:")
print(unique_hosts.take(5))
unique_hosts_count = unique_hosts.count()
print("Unique hosts count: {}".format(unique_hosts_count))

First 5 hosts:
['unicomp6.unicomp.net', '129.94.144.152', 'ppptky391.asahi-net.or.jp', 'slip1.yab.com', 'pm13.j51.com']
Unique hosts count: 81971
```

+ بخش دوم:

با توجه به گنگ بودن صورت سوال این سوال با چند رویکرد حل شد.
در ابتدا تعداد میانگین request‌ها در هر روز محاسبه شد. برای این کار تاریخ و host از داده‌ها جدا شدند و سپس، بر مبنای تاریخ grouping اجرا شد و سپس، بر مبنای تاریخ مرتب‌سازی

صورت گرفت. سپس، تعداد درخواست‌های هر روز بر تعداد host‌های یکتا در هر روز تقسیم شد و میانگین درخواست‌های هر روز حساب شد. برای یکتا کردن host‌ها از محاسبه طول set بر روی این host‌ها استفاده شد. خروجی و کد در زیر قابل مشاهده است.

Average daily requests for each day

```
[ ] daily_hosts = (successful_parsed
    .map(lambda log: (log.date_time.date(), log.host))
    .groupByKey()
    .sortByKey()
    .cache())
print("First 5 daily hosts:")
print(daily_hosts.take(5))

First 5 daily hosts:
[(datetime.date(1995, 7, 1), <pyspark.resultiterable.ResultIterable object at 0x7f37b1654390>), (datetime.date(1995, 7, 2), <pyspark.resultiterable.ResultIterable object at 0x7f37b1654390>), (datetime.date(1995, 7, 3), <pyspark.resultiterable.ResultIterable object at 0x7f37b1654390>), (datetime.date(1995, 7, 4), <pyspark.resultiterable.ResultIterable object at 0x7f37b1654390>), (datetime.date(1995, 7, 5), <pyspark.resultiterable.ResultIterable object at 0x7f37b1654390>)]

[ ] average_daily_requests = (daily_hosts
    .map(lambda dh: (dh[0], len(dh[1])/len(set(dh[1])))))
print("Daily average requests on each host:")
print(average_daily_requests.collect())

Daily average requests on each host:
[(datetime.date(1995, 7, 1), 12.463405238828967), (datetime.date(1995, 7, 2), 12.401728750771763), (datetime.date(1995, 7, 3), 12.207498295841853), (datetime.date(1995, 7, 4), 12.207498295841853), (datetime.date(1995, 7, 5), 12.207498295841853)]
```

در حالت دوم متوسط درخواست‌های هر host محاسبه شده است. برای این کار host و تاریخ جدا شده‌اند و بر مبنای host grouping صورت گرفته است. سپس، تعداد درخواست‌ها به تعداد روزهای فعالیت هر host تقسیم شده است و میانگین درخواست‌های هر host به صورت زیر به دست آمده است.

Average request on each host

```
▶ host_requests = (successful_parsed
    .map(lambda log: (log.host, log.date_time.date())))
    .groupByKey()
    .cache())
print("First 5 host request dates:")
print(host_requests.take(5))

First 5 host request dates:
[('unicomp6.unicomp.net', <pyspark.resultiterable.ResultIterable object at 0x7f37b0c36650>), ('129.94.144.152', <pyspark.resultiterable.ResultIterable object at 0x7f37b0c36650>), ('129.94.144.153', <pyspark.resultiterable.ResultIterable object at 0x7f37b0c36650>), ('129.94.144.154', <pyspark.resultiterable.ResultIterable object at 0x7f37b0c36650>), ('129.94.144.155', <pyspark.resultiterable.ResultIterable object at 0x7f37b0c36650>)]

▶ average_host_daily_requests = (host_requests
    .map(lambda dh: (dh[0], len(dh[1])/len(set(dh[1])))))
print("Daily average requests per host:")
print(average_host_daily_requests.collect())

Daily average requests per host:
[('unicomp6.unicomp.net', 14.0), ('129.94.144.152', 10.8), ('ppptky391.asahi-net.or.jp', 4.0), ('slip1.yab.com', 9.0), ('pm13.j51.com', 3.0), ('129.94.144.153', 10.0), ('129.94.144.154', 10.0), ('129.94.144.155', 10.0)]
```

یک روش دیگر برای این کار استفاده شده است. ابتدا تعداد درخواست‌های هر روز host‌ها محاسبه شده است و سپس host به همراه تعداد درخواست‌ها جدا شده است و بر مبنای host گروه‌بندی شده است و سپس با mapValues مجموع درخواست‌ها به تعداد روزها تقسیم شده است و میانگین درخواست هر host محاسبه شده است. خروجی و کد در زیر قابل مشاهده است.

Number of daily request per host

```
[ ] daily_requests = (successful_parsed
    .map(lambda log: ((log.host, log.date_time.date()), 1))
    .reduceByKey(lambda a, b: a + b))
print("First 5 daily request dates:")
print(daily_requests.take(5))

First 5 daily request dates:
[('205.189.154.54', datetime.date(1995, 7, 1)), 11], ('alyssa.prodigy.com', datetime.date(1995, 7, 1)), 536], ('ix-orl2-01.ix.netcom.com', datetime.date(1995, 7, 1)),
```

Calculate daily average

```
( ) average_daily = (daily_requests
    .map(lambda record: (record[0][0], record[1]))
    .groupByKey()
    .mapValues(lambda reqs: sum(reqs) / len(reqs)))
print("First 5 average daily requests:")
print(average_daily.collect())

First 5 average daily requests:
[('pm13.j51.com', 3.0), ('usr7-dialup46.chicago.mci.net', 1.0), ('wwwproxy.info.au', 27.15), ('204.120.76.119', 2.0), ('199.2.253.2', 1.0), ('slip-774.netaxs.com', 5.0),
```

+ بخش سوم:

برای محاسبه تعداد گیفها باید endpoint با gif. به پایان برسد. با استفاده از این شرط و filter این درخواست‌ها جدا شده‌اند. تعداد این درخواست‌ها با استفاده از count شمرده شده‌است. خروجی و کد در زیر قابل مشاهده است. تعداد این درخواست‌ها ۱۰۲۹۴۱۸ است.

Part 3

```
( ) gif_request = (successful_parsed
    .filter(lambda log: log.endpoint.endswith('.gif')))
print("First 5 gif requests:")
print(gif_request.take(5))
gif_requests_count = gif_request.count()
print("GIF requests count: {}".format(gif_requests_count))

First 5 gif requests:
[Row(host='199.120.110.21', client_identd='-', user_id='-', date_time=datetime.datetime(1995, 7, 1, 0, 0, 11), method='GET', endpoint='/shuttle/missions/sts-73/sts-73-patch-sma
GIF requests count: 1029418
```

+ بخش چهارم:

برای این بخش ابتدا ip‌ها با استفاده از چک کردن عدد نبودن کarakتر اول endpoint حذف شده‌اند. سپس، هر دامنه به یک map شده‌است و بر اساس دامنه reduce شده‌است و تعداد هر دامنه محاسبه شده‌است. سپس، بیش از سه درخواست داشتن فیلتر شده‌است. خروجی دامنه‌ها مرتب شده بر اساس تعداد درخواست‌ها در زیر قابل مشاهده است.

Most requested domains

```
[ ] IP_STARTS = ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"]
domains_request_count = (successful_parsed
    .filter(lambda log: log.endpoint[0] not in IP_STARTS)
    .map(lambda log: (log.endpoint, 1))
    .reduceByKey(lambda a, b: a + b)
    .cache())

top_domains = domains_request_count.takeOrdered(10, lambda record: -1 * record[1])
print("Top 10 domains:")
print(top_domains)

Top 10 domains:
[('/images/NASA-logosmall.gif', 111331), ('/images/KSC-logosmall.gif', 89639), ('/images/MOSAIC-logosmall.gif', 60468), ('/images/USA-logosmall.gif', 60014),
```



```
[ ] more_than_three_domains = (domains_request_count
    .filter(lambda record: record[1]>3))
print("More than 3 requests domains:")
more_three_count = more_than_three_domains.count()
print(more_than_three_domains.takeOrdered(more_three_count, lambda record: -1 * record[1]))
```



```
More than 3 requests domains:
[('/images/NASA-logosmall.gif', 111331), ('/images/KSC-logosmall.gif', 89639), ('/images/MOSAIC-logosmall.gif', 60468), ('/images/USA-logosmall.gif', 60014),
```

شده‌اند. سپس، `grouping`‌ها به همراه تعداد درخواست آن‌ها در هر روز با استفاده از `endpoint` بر اساس تاریخ محاسبه شده‌اند. همچنین، این لیست بر مبنای تعداد مرتب شده‌است. سپس، از هر تاریخ بیشترین تعداد استخراج شده‌است و نمایش داده شده‌است. خروجی و کد به صورت زیر است.

Most requested domains in each day

```
[ ] IP_STARTS = ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"]
daily_domain_requests = (successful_parsed
    .filter(lambda log: log.endpoint[0] not in IP_STARTS)
    .map(lambda log: ((log.date_time.date(), log.endpoint), 1))
    .reduceByKey(lambda a, b : a + b)
    .cache())

top_domains = daily_domain_requests.takeOrdered(10, lambda record: -1 * record[1])
Find and replace > op 10 domains:
print(top_domains)

Top 10 domains:
[((datetime.date(1995, 7, 13), '/images/NASA-logosmall.gif'), 12098), ((datetime.date(1995, 7, 12), '/images/NASA-logosmall.gif'), 7350), ((datetime.date(1995, 7, 13), '/htb

[ ] daily_top_domain = (daily_domain_requests
    .map(lambda log: (log[0][0], (log[1], log[0][1])))
    .groupByKey()
    .map(lambda x : (x[0], sorted(list(x[1]), reverse=True)))
    .cache())

print(daily_top_domain.take(5))

[(datetime.date(1995, 7, 3), [(5573, '/images/NASA-logosmall.gif'), (4655, '/images/KSC-logosmall.gif'), (3666, '/shuttle/countdown/count.gif'), (3534, '/shuttle/countdown/'

top_domains = (daily_top_domain
    .map(lambda day_record: (day_record[0], day_record[1][0])))

print("Daily top domain:")
top_domains.collect()

Daily top domain:
[(datetime.date(1995, 7, 3), (5573, '/images/NASA-logosmall.gif')), (datetime.date(1995, 7, 6), (6227, '/images/NASA-logosmall.gif')), (datetime.date(1995, 7, 9), (1463, '/images/NASA-logosmall.gif')), (datetime.date(1995, 7, 24), (3362, '/images/NASA-logosmall.gif')), (datetime.date(1995, 7, 25), (3380, '/images/NASA-logosmall.gif')), (datetime.date(1995, 7, 28), (1555, '/images/NASA-logosmall.gif')), (datetime.date(1995, 7, 2), (3416, '/images/NASA-logosmall.gif')), (datetime.date(1995, 7, 13), (12098, '/images/NASA-logosmall.gif')), (datetime.date(1995, 7, 14), (4876, '/images/NASA-logosmall.gif')), (datetime.date(1995, 7, 1), (3977, '/images/NASA-logosmall.gif')), (datetime.date(1995, 7, 15), (2411, '/images/KSC-logosmall.gif')), (datetime.date(1995, 7, 16), (2330, '/images/KSC-logosmall.gif')), (datetime.date(1995, 7, 5), (6175, '/images/NASA-logosmall.gif')), (datetime.date(1995, 7, 10), (3740, '/images/NASA-logosmall.gif')), (datetime.date(1995, 7, 11), (5274, '/images/NASA-logosmall.gif')), (datetime.date(1995, 7, 12), (7350, '/images/NASA-logosmall.gif')), (datetime.date(1995, 7, 17), (3949, '/images/NASA-logosmall.gif')), (datetime.date(1995, 7, 19), (3739, '/images/NASA-logosmall.gif')), (datetime.date(1995, 7, 22), (1702, '/images/KSC-logosmall.gif')), (datetime.date(1995, 7, 23), (1699, '/images/KSC-logosmall.gif')), (datetime.date(1995, 7, 27), (3270, '/images/NASA-logosmall.gif')), (datetime.date(1995, 7, 4), (3858, '/images/NASA-logosmall.gif')), (datetime.date(1995, 7, 7), (5055, '/images/NASA-logosmall.gif')), (datetime.date(1995, 7, 8), (1718, '/images/NASA-logosmall.gif')), (datetime.date(1995, 7, 18), (3522, '/images/NASA-logosmall.gif')), (datetime.date(1995, 7, 20), (3545, '/images/NASA-logosmall.gif')), (datetime.date(1995, 7, 21), (3574, '/images/KSC-logosmall.gif')), (datetime.date(1995, 7, 26), (3151, '/images/NASA-logosmall.gif'))]
```

+ بخش پنجم:

برای این بخش، درخواست‌های غیر ۲۰۰ فیلتر شده‌اند و سپس، هر ارور به یک نگاشت داده شده است. سپس، با استفاده از `reduceByKey` تکرار هر کد خطا محاسبه شده است و در نهایت با استفاده از `matplotlib` نمودار میله‌ای آن‌ها رسم شده است. کد و خروجی این بخش در زیر قابل مشاهده است.

Find Error Logs

```
[ ] http_errors = (successful_parsed
                    .filter(lambda log: log.response_code!=200)
                    .map(lambda log: (log.response_code, 1))
                    .cache())
print("First 5 errors:")
print(http_errors.take(5))
```

```
First 5 errors:
[(304, 1), (304, 1), (304, 1), (302, 1), (404, 1)]
```

Count errors

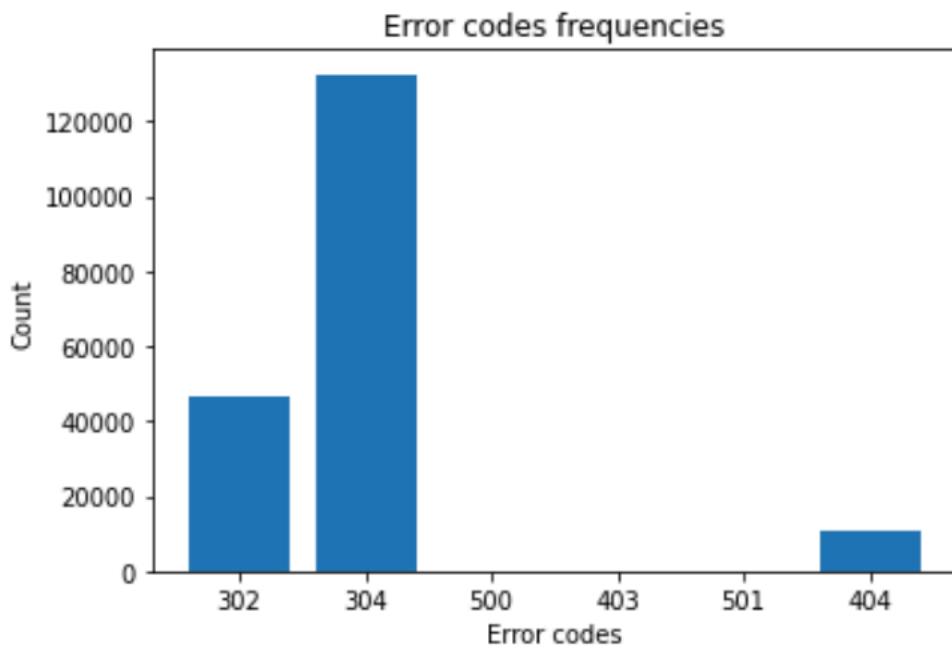
```
[ ] error_counts = (http_errors
                     .reduceByKey(lambda a, b: a + b))
print("Error counts:")
print(error_counts.collect())
```

```
Error counts:
[(302, 46569), (304, 132626), (500, 62), (403, 54), (501, 14), (404, 10784)]
```

Draw status code frequency bar plot

```
status_codes = []
code_frequencies = []
for error, count in error_counts.collect():
    status_codes.append(str(error))
    code_frequencies.append(count)

plt.title("Error codes frequencies")
plt.xlabel("Error codes")
plt.ylabel("Count")
plt.bar(status_codes, code_frequencies)
plt.show()
```



• گام سوم – کار با دیتافریم‌ها / Spark SQL

در ابتدا تنظیمات را مشابه گام قبلی انجام می‌دهیم. تنها تفاوت آن است که چند کتابخانه دیگر نیز import شده‌اند. همچنین داده‌ها به دلیل زیاد و حجمی بودن روی درایو قرار گرفته‌اند و به همین دلیل، نیاز است تا drive import نیز شود تا با آن بتوان درایو را mount کرد و داده‌ها را از آن خواند.

```

import os
import pandas as pd
from google.colab import drive
from tqdm import tqdm

from pyspark.sql.functions import lit, col
from pyspark.sql.types import DoubleType, IntegerType, StringType, LongType
from pyspark.sql import functions as F
from pyspark.sql.window import Window

```

داده‌های دو ماه از بورس حاوی ۴۰ روز کاری(فایل) بر روی درایو اپلود شد و از آن‌ها استفاده شده‌است.

سپس، با استفاده از اسپارک از پوشه `mount` شده داده‌ها را خوانده و سه ستون روز، ماه و سال به داده‌های هر پوشه اضافه شد و به یک دیتافریم فایل به فایل اضافه شد. کد این قسمت در زیر قابل مشاهده است. البته پیش از خوانده شدن توسط اسپارک با استفاده از `pandas` فرمت از `excel` به `CSV` تغییر داده شده است.

Clean & Convert excel files to csv to be compatible with spark

```

stocks_folder_path = '/content/drive/MyDrive/BD Stocks/xlsx'
stocks_csv_folder_path = '/content/drive/MyDrive/BD Stocks/csv'
for file_name in tqdm(os.listdir(stocks_folder_path)):
    file_path = os.path.join(stocks_folder_path, file_name)
    data = pd.read_excel(file_path)
    header = data.iloc[1]
    data = data.iloc[2:]
    data.columns = header
    csv_file_name = file_name.split(".")[0] + ".csv"
    csv_path = os.path.join(stocks_csv_folder_path, csv_file_name)
    data.to_csv(csv_path)

```

⌚ 100% | [██████████] | 40/40 [00:05<00:00, 6.94it/s]

```

is_first = True
for file_name in tqdm(os.listdir(stocks_csv_folder_path)):
    file_path = os.path.join(stocks_csv_folder_path, file_name)
    dataframe = spark.read.csv(file_path, header=True, sep=",")
    year, month, day = file_name.split('.')[0].split('_')[2:]
    dataframe = dataframe.withColumn("day", lit(day))
    dataframe = dataframe.withColumn("month", lit(month))
    dataframe = dataframe.withColumn("year", lit(year))
    dataframe = dataframe.withColumn('بیشترین', F.col('').cast(IntegerType()))
    dataframe = dataframe.withColumn('کمترین', F.col('').cast(IntegerType()))
    dataframe = dataframe.withColumn('قیمت پایانی - درصد', F.col('').cast(DoubleType()))
    dataframe = dataframe.withColumn('قیمت پایانی - تغییر', F.col('').cast(DoubleType()))
    dataframe = dataframe.withColumn('قیمت پایانی - مقدار', F.col('').cast(IntegerType()))
    dataframe = dataframe.withColumn('آخرین معامله - درصد', F.col('').cast(DoubleType()))
    dataframe = dataframe.withColumn('آخرین معامله - تغییر', F.col('').cast(DoubleType()))
    dataframe = dataframe.withColumn('آخرین معامله - مقدار', F.col('').cast(IntegerType()))
    dataframe = dataframe.withColumn('اولین', F.col('').cast(IntegerType()))
    dataframe = dataframe.withColumn('دیروز', F.col('').cast(IntegerType()))
    dataframe = dataframe.withColumn('ارزش', F.col('').cast(LongType()))
    dataframe = dataframe.withColumn('حجم', F.col('').cast(LongType()))
    dataframe = dataframe.withColumn('تعداد', F.col('').cast(LongType()))
    dataframe = dataframe.withColumnRenamed('بیشترین', 'max_price') \
        .withColumnRenamed('کمترین', 'min_price') \
        .withColumnRenamed('قیمت پایانی - درصد', 'close_price_change_percent') \
        .withColumnRenamed('قیمت پایانی - تغییر', 'close_price_change') \
        .withColumnRenamed('قیمت پایانی - مقدار', 'close_price') \
        .withColumnRenamed('آخرین معامله - درصد', 'last_order_value_change_percent') \
        .withColumnRenamed('آخرین معامله - تغییر', 'last_order_value_change') \
        .withColumnRenamed('آخرین معامله - مقدار', 'last_order_value') \
        .withColumnRenamed('اولین', 'first_order_value') \
        .withColumnRenamed('ارزش', 'value') \
        .withColumnRenamed('دیروز', 'yesterday_qnt') \
        .withColumnRenamed('حجم', 'volume') \
        .withColumnRenamed('تعداد', 'quantity') \
        .withColumnRenamed('نام', 'full_name') \
        .withColumnRenamed('_', 'symbol')

    dataframe = dataframe.drop('_c0')
    if is_first:
        final_df = dataframe
        is_first = False
    else:
        final_df = final_df.union(dataframe)
print(f'Final dataframe rows count: {final_df.cache().count()}')
print('First 5 rows:')
final_df.show(5)

```

```

100% [██████████] 40/40 [00:10<00:00,  3.77it/s]
Final dataframe rows count: 35213
First 5 rows:
+-----+-----+-----+-----+-----+-----+-----+-----+
|symbol| full_name|quantity| volume| value|yesterday_qnt|first_order_value|last_order_value|last_order_value_change|
+-----+-----+-----+-----+-----+-----+-----+-----+
|1399|11 |1 |18400 |17010 |2.16- |387.0- |17518 |2.9- |520.0- |17385
|1399|11 |1 |51716 |46792 |1.8 |885.0 |50139 |5.0 |2462.0 |5171
|1399|11 |1 |38707 |35028 |0.85 |314.0 |37178 |5.0 |1843.0 |38707
|1399|11 |1 |47395 |47395 |0.84 |379.0 |45518 |5.0 |2256.0 |47395
|1399|11 |1 |10920 |10502 |1.49 |156.0 |10657 |1.06 |111.0 |10612
+-----+-----+-----+-----+-----+-----+-----+-----+
only showing top 5 rows

```

از جمله تغییرات دیگری که داده شد تغییر فرمت ستون‌ها و تغییر نام آن‌ها به انگلیسی برای راحت‌تر پیداکردن و استفاده از آن‌ها بود.
خروجی نمونه‌ای در بالا قابل مشاهده است.

برای استفاده از قابلیت‌های Spark SQL نیاز است تا دیتابریم ایجادشده را به عنوان یک جدول register کنیم تا بتوانیم روی آن کوئری SQL بزنیم.

Make temporary table for sql queries

```
[ ] final_df.registerTempTable('stocks')
```

برای هر بخش کد و خروجی آن در زیر قابل مشاهده است.
+ بخش اول:

ابتدا باید داده‌ی مربوط به روز آخر را برای این سوال از داده‌ها کل استخراج کنیم.

Extract last day dataframe

```
[ ] last_day_df = final_df.filter((final_df['day']==27) &
                                 (final_df['month']==12) &
                                 (final_df['year']==1399))
last_day_df.cache().show(5)
print(f"Final day record count: {last_day_df.count()}")
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
|symbol| full_name|quantity| volume| value|yesterday_qnt|first_order_value|last_order_value|last_order_value_change|
+-----+-----+-----+-----+-----+-----+-----+-----+
|1399|12 |27 |87650 |84010 |0.72- |620.0- |85740 |1.98- |1710.0-
|1399|12 |27 |1 |1 |0.0 |0.0 |1 |0.0 |0.0
|1399|12 |27 |95683 |88462 |2.22 |2006.0 |92273 |6.0 |5416.0
|1399|12 |27 |14990 |14330 |3.46 |490.0 |14670 |2.89 |410.0
|1399|12 |27 |18100 |17720 |1.0- |180.0- |17900 |0.61- |110.0-
+-----+-----+-----+-----+-----+-----+-----+-----+
only showing top 5 rows
```

Final day record count: 934

با استفاده از دیتابریم و پیدا کردن گرانترین:
برای این کار نیاز است تا دیتابریم را بر اساس قیمت پایانی به صورت نزولی مرتب کنیم و ۱۰ داده آخر را انتخاب کنیم و نماد و قیمت پایانی را خروجی دهیم.

Most expensive

```
[ ] most_expensive = last_day_df.orderBy('close_price', ascending=False).limit(10)
print("10 Most expensive symbols:")
most_expensive.select(most_expensive['symbol'], most_expensive['close_price']).show()
```

symbol	close_price
1484467	سنت ۰۰۹
1398458	پست ۰۰۸
1372579	پست ۰۰۸
1362991	پست ۰۰۸
1350000	پست ۰۰۸
1105423	افاد ۱۴
1100000	صگل ۳۰۹
1090808	سکه ۰۲
1090417	سکه ۹۹۱۲
1089400	سکه ۰۰۱۲

با استفاده از دیتافریم و پیدا کردن ارزانترین:
رای این کار نیاز است تا دیتافریم را بر اساس قیمت پایانی به صورت صعودی مرتب کنیم و ۱۰ داده آخر را انتخاب کنیم و نماد و قیمت پایانی را خروجی دهیم.

Cheapest

```
[ ] most_expensive = last_day_df.orderBy('close_price').limit(10)
print("10 Cheapest symbols:")
most_expensive.select(most_expensive['symbol'], most_expensive['close_price']).show()
```

symbol	close_price
1	کیان ۲
1	امین یکم ۲
1	باقوت ۲
1	حامی ۱۴۰۱
1	کارین ۲
1	آکورد ۲
1	کمند ۲
1	ویازار ۲
1	نهال ۲
1	اعتماد ۲

با استفاده از SQL و پیدا کردن گرانترین:
برای این خواسته باید با استفاده از دستور `order by` جدول را بر اساس قیمت پایانی به صورت نزولی مرتب کنیم و ۱۰ داده آخر را با `limit` انتخاب کنیم و نماد و قیمت پایانی را خروجی دهیم.

Most Expensive

```
▶ print("10 Most expensive symbols:")
spark.sql("SELECT symbol, close_price " +
          "FROM stocks " +
          "WHERE (year == 1399 AND month==12 AND day==27) " +
          "ORDER BY close_price DESC " +
          "LIMIT 10;").show()
```

⇨ 10 Most expensive symbols:

symbol	close_price
1484467	سنفت ۰۹
1398458	پست ۰۹۰۰۰۸
1372579	پست ۰۸۰۰۰۸
1362991	پست ۰۴۰۰۰۸
1350000	پست ۰۲۰۰۰۸
1105423	افاد ۱۴
1100000	صگل ۳۰۹
1090808	سکه ۰۲۰۰۱۱
1090417	سکه ۰۴۹۹۱۲
1089400	سکه ۰۴۰۰۱۲

با استفاده از SQL و پیدا کردن ارزانترین:
برای این خواسته باید با استفاده از دستور order by جدول را بر اساس قیمت پایانی به صورت
 سعودی مرتب کنیم و ۱۰ داده آخر را با limit انتخاب کنیم و نماد و قیمت پایانی را خروجی
 دهیم.

Cheapest

```
▶ print("10 Cheapest symbols:")
spark.sql("SELECT symbol, close_price " +
          "FROM stocks " +
          "WHERE (year == 1399 AND month==12 AND day==27) " +
          "ORDER BY close_price ASC " +
          "LIMIT 10;").show()
```

```
⇨ 10 Cheapest symbols:
+-----+-----+
|   symbol|close_price|
+-----+-----+
|       1 | کیان 2 |
|       1 | امین یکم 2 |
|       1 | یاقوت 2 |
|       1 | حامی 1401 |
|       1 | کارین 2 |
|       1 | آکورد 2 |
|       1 | کمند 2 |
|       1 | وبازار 2 |
|       1 | نهال 2 |
|       1 | اعتماد 2 |
+-----+-----+
```

+ بخش دوم:

با استفاده از دیتافریم:

برای این منظور باید نمادها را بر اساس حجم به صورت نزولی مرتب کرد و اولین آن‌ها را خروجی داد.

With Dataframe

```
▶ most_traded = final_df.orderBy('volume', ascending=False).limit(1)
print("Most traded symbol:")
most_traded.select(most_traded['symbol'], most_traded['volume']).show()
```

```
⇨ Most traded symbol:
+-----+-----+
| symbol|      volume|
+-----+-----+
| 34152999908 | 4رسان |
+-----+-----+
```

با استفاده از Spark SQL

برای این منظور باید نمادها را بر اساس حجم به صورت نزولی مرتب کرد(با order by) و اولین آن‌ها را خروجی داد(limit).

With Spark SQL

```
▶ print("Most traded symbol:")
spark.sql("SELECT symbol, volume " +
          "FROM stocks " +
          "ORDER BY volume DESC " +
          "LIMIT 1;").show()
```

```
⇨ Most traded symbol:
+-----+-----+
| symbol |      volume |
+-----+-----+
| 34152999908 | 42954 |
+-----+-----+
```

+ بخش سوم:

با استفاده از دیتافریم:

برای این منظور بر مبنای ماه و نماد گروهبندی می‌کنیم و سپس، مجموع تغییرات قیمت را با هم جمع می‌زنیم و نام آن‌ها را تغییر می‌دهیم. سپس با استفاده از partition بر اساس ماه و مرتب سازی بر اساس تغییر ماه به صورت نزولی عملیات ranking رخ می‌دهد. سپس، با استفاده از فیلتر ۱۰ رتبه اول هر ماه انتخاب شده‌است.

With Dataframe

```
▶ symbol_month_change = (final_df
    .groupBy(['symbol', 'month'])
    .agg(F.sum('close_price_change').alias('month_change')))

window_spec = Window().partitionBy(['month']).orderBy(F.desc('month_change'))
ranked_symbols = symbol_month_change.withColumn("rank", F.rank().over(window_spec))
print("Most price rised symbol in each month:")
most_rised = (ranked_symbols
    .filter(ranked_symbols['rank'] < 11))
most_rised.show()
```

⇒ Most price rised symbol in each month:

	symbol	month	month_change	rank
1	339281.0	11	805	تملی
2	188905.0	11	01پ0001	کشم
3	166519.0	11	03پ0112	سکه
4	163059.0	11	01پ0012	سکه
5	161819.0	11	04پ9912	سکه
6	161189.0	11	02پ0011	سکه
7	147381.0	11	02پ0008	پست
8	145293.0	11	134	اراد
9	139102.0	11	2	عکاوه
10	113860.0	11	9809	تسه
1	348744.0	12	14	افاد
2	329504.0	12	803	تملی
3	285136.0	12	806	تملی
4	236888.0	12	804	تملی
5	179685.0	12	802	تملی
6	160836.0	12	24	افاد
7	149670.0	12	703	تملی
8	135483.0	12	اعلی جم	اعلی جم
9	129999.0	12	02پ0006	کشم
10	112789.0	12	844	افاد

با استفاده از Spark SQL

برای این منظور بر مبنای ماه و نماد گروهبندی می‌کنیم و سپس، مجموع تغییرات قیمت را با هم جمع می‌زنیم و نام آن‌ها را تغییر می‌دهیم و در جدول symbol_month_change می‌گذاریم. سپس با استفاده از partition بر اساس ماه و مرتب سازی بر اساس تغییر ماه به صورت نزولی عملیات ranking رخ می‌دهد و در جدول ranked_symbols قرار داده می‌شود. سپس، با استفاده از فیلتر ۱۰ رتبه اول هر ماه از این جدول انتخاب می‌شود.

With Spark SQL

```
[ ] print("Most price rised symbol in each month:")
spark.sql("SELECT * " +
    "FROM ( " +
    " SELECT symbol, month, month_change, " +
    " row_number() over (partition by month order by month_change desc) as symbol_rank " +
    " FROM ( " +
    "     SELECT symbol, month, SUM(close_price_change) as month_change " +
    "     FROM stocks " +
    "     GROUP BY symbol, month " +
    " ) AS symbol_month_change " +
    " ) AS ranked_symbols " +
    "WHERE symbol_rank <= 10;").show()
```

Most price rised symbol in each month:			
	symbol	month	month_change
			symbol_rank
1	339281.0	11	805
2	188905.0	11	01.پ0001
3	166519.0	11	03.پ0112
4	163059.0	11	01.پ0012
5	161819.0	11	04.پ9912
6	161189.0	11	02.پ0011
7	147381.0	11	02.پ0008
8	145293.0	11	134
9	139102.0	11	2
10	113860.0	11	9809
1	348744.0	12	14
2	329504.0	12	803
3	285136.0	12	806
4	236888.0	12	804
5	179685.0	12	802
6	160836.0	12	24
7	149670.0	12	703
8	135483.0	12	عبلی جم
9	129999.0	12	کشم 0006
10	112789.0	12	افداد 844

+ بخش چهارم: با استفاده از دیتافریم:

برای این منظور بر مبنای نماد گروهبندی می‌کنیم و سپس، مجموع تغییرات قیمت را با هم جمع می‌زنیم و نام آن‌ها را تغییر می‌دهیم. سپس بر مبنای این ویژگی جدید مرتب سازی را انجام می‌دهیم و ۱۰ داده‌ی آخر را بر می‌گردانیم. لازم به ذکر است به دلیل اینکه داده‌ی دو ماه را داریم نیاز به فیلتر روی تاریخ تا شش ماه پیش نیست.

```

▶ most_fall = (final_df
                .groupBy('symbol')
                .agg(F.sum('close_price_change').alias('six_month_change'))
                .sort('six_month_change')).limit(10)
print("Most price fall symbol in 6 month:")
most_fall.show()

```

⇨ Most price fall symbol in 6 month:

symbol	six_month_change
7030216.0-	2میلس
1950368.0-	اراد 314
1807954.0-	تسه 99102
1771084.0-	تسه 99092
1769006.0-	آگا 2میلس
1754240.0-	تسه 99082
1422494.0-	اراد 344
1309024.0-	اراد 424
1276771.0-	اراد 494
1210055.0-	اراد 384

با استفاده از Spark SQL

برای این منظور بر مبنای نماد گروهبندی می‌کنیم و سپس، مجموع تغییرات قیمت را با هم جمع می‌زنیم(Sum) و نام آن‌ها را تغییر می‌دهیم(as). سپس بر مبنای این ویژگی جدید مرتب سازی را انجام می‌دهیم(order by) و ۱۰ داده‌ی آخر را برابر می‌گردانیم(limit). لازم به ذکر است به دلیل اینکه داده‌ی دو ماه را داریم نیاز به فیلتر روی تاریخ تا شش ماه پیش نیست.

With Spark SQL

```

▶ print("Most price fall symbol in 6 month:")
spark.sql("SELECT symbol, SUM(close_price_change) as period_change " +
          "FROM stocks " +
          "GROUP BY symbol " +
          "ORDER BY period_change ASC " +
          "LIMIT 10;").show()

```

⇨ Most price fall symbol in 6 month:

symbol	period_change
7030216.0-	2میلس
1950368.0-	اراد 314
1807954.0-	تسه 99102
1771084.0-	تسه 99092
1769006.0-	آگا 2میلس
1754240.0-	تسه 99082
1422494.0-	اراد 344
1309024.0-	اراد 424
1276771.0-	اراد 494
1210055.0-	اراد 384

+ بخش پنجم:

با استفاده از دیتافریم:

برای به دست آوردن این که به چه میزانی نمادی بسته بوده است تعداد روزهای باز بودن آن را باید محاسبه کنیم و بر مبنای صعودی مرتب کنیم. نمادی با کمتری میزان باز بودن بیشترین میزان بسته بودن را داشته است.

برای این منظور بر مبنای نماد گروه بندی انجام می دهیم و سپس، می شماریم و بر مبنای count مرتب می کنیم. این گروه بندی به این منظور است که رکوردهای نماد در کنار هم قرار بگیرند.

With Dataframe

```
[ ] most_closed = final_df.groupBy('symbol').count().orderBy('count')
print("Most closed symbols:")
most_closed.show()
```

```
Most closed symbols:
+-----+----+
| symbol | count |
+-----+----+
| 1      | 4زد   |
| 1      | 99112  |
| 1      | 58     |
| 1      | 2011   |
| 1      | 2032   |
| 1      | 4آسام  |
| 1      | 18     |
| 1      | 7182   |
| 1      | 2سدشت |
| 1      | 3092   |
| 1      | 4ملت   |
| 1      | 4مداران |
| 1      | 1125   |
| 1      | 808    |
| 1      | 3001   |
| 1      | 09     |
| 1      | 4122   |
| 1      | 37     |
| 1      | 2وتعاون |
| 1      | 55     |
+-----+----+
only showing top 20 rows
```

با استفاده از Spark SQL

برای به دست آوردن این که به چه میزانی نمادی بسته بوده است تعداد روزهای باز بودن آن را باید محاسبه کنیم و بر مبنای صعودی مرتب کنیم. نمادی با کمتری میزان باز بودن بیشترین میزان بسته بودن را داشته است.

برای این منظور بر مبنای نماد گروهبندی انجام می‌دهیم و سپس، می‌شماریم و بر مبنای count مرتب می‌کنیم. این گروهبندی به این منظور است که رکوردهای نماد در کنار هم قرار بگیرند.

With Spark SQL

```
▶ print("Most closed symbols:")
spark.sql("SELECT symbol, COUNT(*) as open_days " +
          "FROM stocks " +
          "GROUP BY symbol " +
          "ORDER BY open_days ASC;").show()
```

```
→ Most closed symbols:
+-----+-----+
|   symbol |open_days |
+-----+-----+
|       1   |دروز 4|
|       1   |آسام 4|
|       1   |اراد 58|
|       1   |صکل 3092|
|       1   |تسه 99112|
|      1    |ساپا 2032|
|       1   |پا ر 18|
|       1   |اخز 7182|
|       1   |وتعاون 2|
|       1   |سدشت 2|
|       1   |مداران 4|
|       1   |اراد 37|
|       1   |افاد 55|
|      1    |پسان 2011|
|       1   |ملت 4|
|       1   |تملي 808|
|       1   |ضفار 1125|
|       1   |ضغدر 3001|
|      1    |صمعاد 4122|
|       1   |اروند 09|
+-----+-----+
only showing top 20 rows
```

تعداد بیشتری نماد در این بخش خروجی داده شده است چرا که تعداد زیادی ۱ روز باز بودن وجود داشته است که بیشترین میزان بسته بودن را داشته اند.

• گام چهارم – Spark GraphX

در ابتدا تنظیمات را مشابه گام قبلی انجام می‌دهیم.

تنها تفاوت آن است که چند کتابخانه دیگر نیز import شده‌اند. همچنین داده‌ها به دلیل زیاد و حجمی بودن روی درایو قرار گرفته‌اند و به همین دلیل، نیاز است تا drive نیز import شود تا با آن بتوان درایو را mount کرد و داده‌ها را از آن خواند.
به دلیل عدم وجود درایور مناسب برای graphframes آخرین نسخه نیاز شد تا اسپارک را به نسخه پایین تر منتقل کرده و چند متغیر محیطی دیگر را نیز اضافه کنیم.

```
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
!wget -q http://archive.apache.org/dist/spark/spark-2.4.8/spark-2.4.8-bin-hadoop2.6.tgz
!tar xf spark-2.4.8-bin-hadoop2.6.tgz
!pip -q install findspark graphframes
```

: environment variables

```
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-2.4.8-bin-hadoop2.6"

os.environ["HADOOP_HOME"] = os.environ["SPARK_HOME"]
os.environ["PYSPARK_DRIVER_PYTHON"] = "jupyter"
os.environ["PYSPARK_DRIVER_PYTHON_OPTS"] = "notebook"
os.environ["PYSPARK_SUBMIT_ARGS"] = "--packages graphframes:graphframes:0.8.0-spark2.4-s_2.11 pyspark-shell"
```

همچنین باید یک آرگومان برای pypark تعريف شود تا بتواند از graphframes استفاده کند. import Graphframes نیز باید باشد.

+ بخش اول:

در این بخش فایل گره‌ها و یال‌ها را از درایو می‌خوانیم، آن‌ها را در دو لیست می‌ریزیم و فایل‌ها را می‌بندیم. سپس، با استفاده spark.createDataframes دیتافریم‌های مناسب را درست می‌کنیم. سپس، با استفاده از این دو دیتافریم یک GraphFrame می‌سازیم که ترکیبی از spark graphx و دیتافریم است.

در زیر کد و خروجی شما دو دیتافریم اولیه را می‌بینیم. همچنین، graph frame‌ای با نام wiki_graph نیز ایجاد شده است.

```

vertices_data = spark.createDataFrame(vertices, ["id", "title"])
print("Vertices Schema:")
vertices_data.printSchema()

edges_data = spark.createDataFrame(edges, ["src", "dst"])
print("Edges Schema:")
edges_data.printSchema()

wiki_graph = GraphFrame(vertices_data, edges_data)

```

```

Vertices Schema:
root
|-- id: string (nullable = true)
|-- title: string (nullable = true)
```

```

Edges Schema:
root
|-- src: string (nullable = true)
|-- dst: string (nullable = true)
```

Show graph vertices and edges

```

▶ wiki_graph.vertices.show()
wiki_graph.edges.show()
```

	id	title
8774773382640701231	List of Canadian ...	
7761101435592530731	List of Football ...	
8133629125642450577	Template:2004 Foo...	
7698287352205296646	File:North Melbou...	
2097725138520422255	Georgia Football ...	
6922631041164063688	Football at the 2...	
1197038737428485418	Wikipedia:Miscell...	
2558316557265591251	Football at the 2...	
1775630591844818068	Serbian Football ...	
4205664098526595262	Wikipedia:WikiPro...	
1033687574081081174	1943–44 Netherlan...	
6862802626977892137	1948–49 Football ...	
3681282860489534792	1998–99 North Wes...	
8747320282862100730	1984–85 Football ...	
7895242920481704787	Template:1976–77 ...	
4200215839332270980	Black Thunder Rug...	
687511193117412416	2008 UEFA Europea...	
6058169767624236103	Template:College ...	
1244764815707993800	Template:1976–77 ...	
5775454784407389519	Football at the 2...	

only showing top 20 rows

	src	dst
3205189634386258258	5457500977524455425	
7434181864825576211	3992155592538800813	
7434181864825576211	5174708811275211225	
7434181864825576211	8966872558455706256	
1138969658277506929	5505576703264774438	
1138969658277506929	1164529579716212743	
2222343049558298343	1053024296822917414	
8566059941063208536	3900444673608155009	

همچنین در بالا چند سطري از هر کدام از `dataframe`ها نمايش داده شده است.
+ بخش دوم:

این گراف دارای یک دیتافریم `inDegrees` است که بیانگر درجه های ورودی گره هاست. با مرتب سازی نزولی این داده ها می توان گره ها با بیشترین درجات را به دست آورد. با `limit` یک داده بیشترین درجه ورودی به دست می آید. سپس، با استفاده از `id` این راس `title` آن را از مجموعه ی گره های این گراف استخراج می کنیم. بیشترین درجه ورودی طبق خروجی زیر ۳۲۷ است.

Max in degree in graph

```
▶ print("Sorted in degrees:")
sorted_in_degree = wiki_graph.inDegrees.sort("inDegree", ascending=False)
sorted_in_degree.show()

print("Highest in degree:")
most_in_degree = sorted_in_degree.limit(1)
most_in_degree.show()
most_in_degree_vertex = (wiki_graph.vertices
                          .filter(f"id == {most_in_degree.collect()[0].id}"))
most_in_degree_vertex.show()
```

```
⇨ Sorted in degrees:
+-----+-----+
|       id|inDegree|
+-----+-----+
| 946065507707541358|    327|
| 3856212023725725593|    322|
| 8978262722425160811|    316|
| 6245498229508734555|    185|
| 7264519433548233535|    180|
| 5362090331808156011|    179|
| 277710621679830671|    149|
| 1984578398767042266|    145|
| 2395551540800395672|    134|
| 5395033957924805072|    130|
| 4254821691068011447|    119|
| 7050959159375889025|    116|
| 3282427710539568335|    115|
| 4512002392249809415|    115|
| 4254821691068016541|    113|
| 4254821691068027845|    107|
| 6623304200448015171|    105|
| 2625575280669536231|    104|
| 6908101889982380382|    102|
| 330493081995023431|    102|
+-----+-----+
only showing top 20 rows
```

Highest in degree:

	id	inDegree
	946065507707541358	327

	id	title
	946065507707541358	The Football League

بیشترین درجه خروجی نیز به صورت مشابه با همین کار روی دیتا فریم outDegrees این گراف به دست می آید. بیشترین درجه خروجی طبق خروجی زیر ۲۶۴ است.

Highest out degree:

	id	outDegree
	3841755165517709241	264

	id	title
	3841755165517709241	Template:All-Irel...

Title: Template:All-Ireland Senior Football Championship

Max out degree in graph

```
▶ print("Sorted in degrees:")
sorted_out_degree = wiki_graph.outDegrees.sort("outDegree", ascending=False)
sorted_out_degree.show()

print("Highest out degree:")
most_out_degree = sorted_out_degree.limit(1)
most_out_degree.show()
most_out_degree_vertex = (wiki_graph.vertices
                           .filter(f"id == {most_out_degree.collect()[0].id}"))
most_out_degree_vertex.show()
print("Title: {}".format(most_out_degree_vertex.collect()[0].title))
```

```
⇨ Sorted in degrees:
+-----+-----+
| id | outDegree |
+-----+-----+
| 3841755165517709241 | 264 |
| 4768697715794291382 | 167 |
| 946065507707541358 | 141 |
| 1984578398767042266 | 132 |
| 1749892575156253660 | 130 |
| 8005203204779397051 | 126 |
| 3794347177295221142 | 125 |
| 5686723817617940514 | 124 |
| 7927626332375316361 | 116 |
| 472311543090593619 | 114 |
| 6245498229508734555 | 110 |
| 5005974931152269142 | 97 |
| 4884850602875197735 | 93 |
| 3282427710539568335 | 92 |
| 7829120379339412244 | 88 |
| 7606187671156652264 | 84 |
| 8473325610216524301 | 80 |
| 6613148896255681939 | 78 |
| 7167801971510974146 | 77 |
| 8050287096245126179 | 72 |
+-----+-----+
only showing top 20 rows
```

+ بخش سوم:

این گراف دارای یک تابع `connectComponents` است که بخش‌های همبندی را محاسبه می‌کند. این تابع بسیار به طول می‌انجامد. به همین دلیل نیاز دارد تا برای آن دایرکتوری گرفتن و قرار دادن چک پوینت فراهم شود. این کار توسط `sparkContext` انجام می‌شود. خروجی زیر نشان‌دهنده خروجی این عملیات است.

در این خروجی به دیتافریم گره‌ها یک ستون `component` اضافه شده است که بیانگر شناسه آن‌هاست.

id	title	component
8774773382640701231	List of Canadian ...	1
7761101435592530731	List of Football ...	1
8133629125642450577	Template:2004 Foo...	1
7698287352205296646	File:North Melbou...	1
2097725138520422255	Georgia Football ...	103079215111
6922631041164063688	Football at the 2...	1
1197038737428485418	Wikipedia:Miscell...	137438953474
2558316557265591251	Football at the 2...	1
1775630591844818068	Serbian Football ...	1
4205664098526595262	Wikipedia:WikiPro...	1
1033687574081081174	1943–44 Netherlan...	8589934631
6862802626977892137	1948–49 Football ...	1
3681282860489534792	1998–99 North Wes...	1
8747320282862100730	1984–85 Football ...	1
7895242920481704787	Template:1976–77 ...	317827579939
4200215839332270980	Black Thunder Rug...	1228360646673
687511193117412416	2008 UEFA Europea...	1
6058169767624236103	Template:College ...	1
1244764815707993800	Template:1976–77 ...	1151051235328
5775454784407389519	Football at the 2...	21

only showing top 20 rows

برای محاسبه سایز هر component کافی است روی شناسه component گروه بندی کنیم و بشماریم. عملیات cache برای بالاتر بردن سرعت در عملیات‌های بعدی انجام شده است. کد و خروجی در زیر قابل مشاهده است.

Calculate size of connected components

```
▶ connected_components_size = (wiki_graph
                                .connectedComponents()
                                .groupBy('component')
                                .count()
                                .cache())
    print("Connected components size:")
    connected_components_size.show()
```

⇨ Connected components size:

component	count
412316860436	1
146028888066	1
146028888086	1
1468878815268	1
1614907703302	1
377957122049	1
884763263008	1
429496729640	1
936302870530	1
1228360646673	1
1511828488192	1
1666447310862	1
506806140928	1
266287972355	1
1382979469334	1
644245094425	1
51539607590	1
1005022347287	1
223338299414	1
17179869191	1

only showing top 20 rows

برای به دست آوردن بزرگترین آنها کافی است آنها را به صورت نزولی مرتب کنیم. کد و خروجی به صورت زیر است.

Showing biggest connect components

```
▶ connected_components_size.sort('count', ascending=False).show()
```

component	count
1	6477
42949672979	52
85899345921	35
8589934631	14
163208757270	14
317827579922	13
77309411351	13
68719476736	12
68719476765	11
34359738381	9
154618822695	7
17179869223	7
687194767374	6
137438953508	6
292057776177	6
420906795013	6
146028888100	6
446676598817	6
730144440335	5
1108101562411	5

only showing top 20 rows

+ بخش چهارم:

برای این کار کافی است دیتافریم vertices را با دیتافریم inDegrees جوین کنیم تا شناسه، درجه ورودی هر گره را در یک جدول کنار هم داشته باشیم.

Join in degrees with vertices data

```
▶ vertices_full = wiki_graph.vertices.join(wiki_graph.inDegrees, 'id').cache()  
print("Vertices data with in degrees:")  
vertices_full.show()
```

⇨ Vertices data with in degrees:

	id	title	inDegree
1	1266831434364880157	National Football...	6
2	1331972860880417523	2010 United Footb...	2
3	1369498455317078482	1938–39 Southern ...	2
4	2000467072865855339	Football at the 1...	2
5	2408272437532239374	1962 College Foot...	3
6	2715722278057256350	Irish Football As...	34
7	3451104515148607245	2009 Adelaide Foo...	2
8	3502541721868043443	2005 Atlantic Ind...	2
9	3920820843608643779	1990 UEFA Europea...	5
10	4441168192505953100	Scotch Old Colleg...	1
11	4499334646599551727	1943 College Foot...	3
12	4680877142281628492	Football at the 2...	2
13	4976384618951573351	Football at the 2...	4
14	5423138579624850276	Devon County Foot...	1
15	5433576273465536249	Hightett Football ...	1
16	585583653337377412	1968–69 Western F...	3
17	6078617804064329033	1954–55 Football ...	4
18	6778984543777808146	2002–03 North Wes...	2
19	7733965970274275986	2011 Swedish Foot...	3
20	8126667952909652252	Football at the 2...	1

only showing top 20 rows

حال روی دیتافریم تازه‌ی بهدست آمده بر مبنای درجه ورودی به صورت نزولی مرتب سازی کنیم و ۱۰تای برتر را با استفاده از limit انتخاب کرده و خروجی دهیم. کد و خروجی در زیر قابل مشاهده است.

همچنین، به دلیل چاپ جدولی به طور کامل عناوین مشخص نیستند که آن‌ها نیز جدا در قالب لیست چاپ شده‌اند.

Find top 10 articles

```
[ ] top_10_articles = (vertices_full
    .sort('inDegree', ascending=False)
    .limit(10))
print("Top 10 articles:")
top_10_articles.show()
print("Full titles:")
print([article.title for article in top_10_articles.collect()])

Top 10 articles:
+-----+-----+
| id | title | inDegree |
+-----+-----+
| 946065507707541358 | The Football League | 327 |
| 3856212023725725593 | National Football... | 322 |
| 897826722425160811 | Australian Footba... | 316 |
| 6245498229508734555 | Southern Football... | 185 |
| 7264519433548233535 | Football League F... | 180 |
| 5362090331808156011 | Football League S... | 179 |
| 277710621679830671 | All-Ireland Senio... | 149 |
| 1984578398767042266 | Scottish Football... | 145 |
| 2395551540800395672 | Football League T... | 134 |
| 5395033957924805072 | Pro Football Hall... | 130 |
+-----+-----+

Full titles:
['The Football League', 'National Football League', 'Australian Football League', 'Southern Football League', 'Football League First Division', 'Football League Second Division']
```

+ بخش پنجم:

مجموعه spark ابزاری برای نمایش گراف به صورت بصری ندارد. به همین منظور، از کتابخانه‌های pyvis و networkx برای نمایش گراف استفاده شده است.

برای این کار ابتدا با استفاده از کتابخانه networkx دیتافریم را به گرافی مناسب نمایش تبدیل می‌کنیم. سپس، با استفاده از networkViz از کتابخانه dimcli که بسطی از networkViz است، این گراف را نمایش می‌دهیم. کد و نتیجه در زیر قابل مشاهده است.

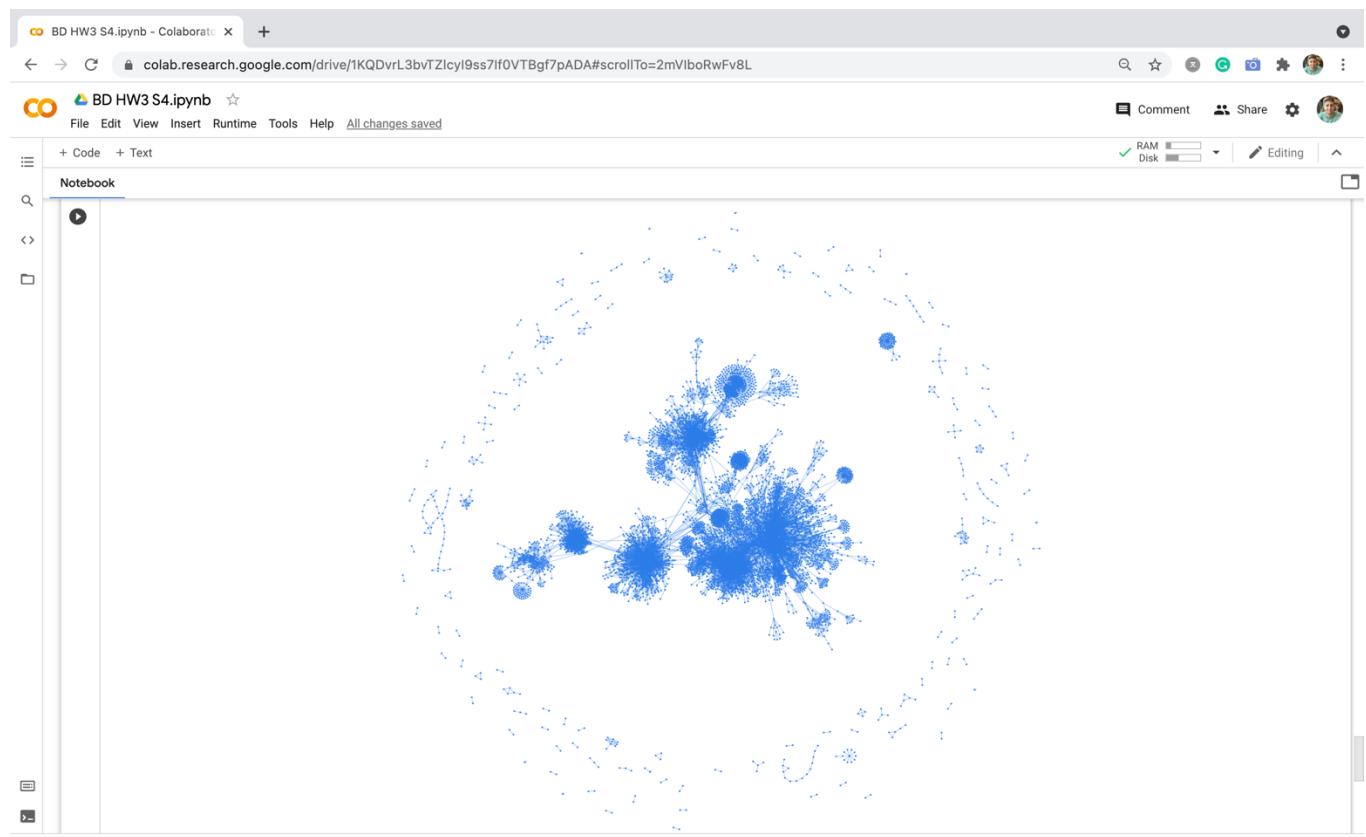
Install requirements

```
▶ pip3 install -q networkx dimcli pyvis
```

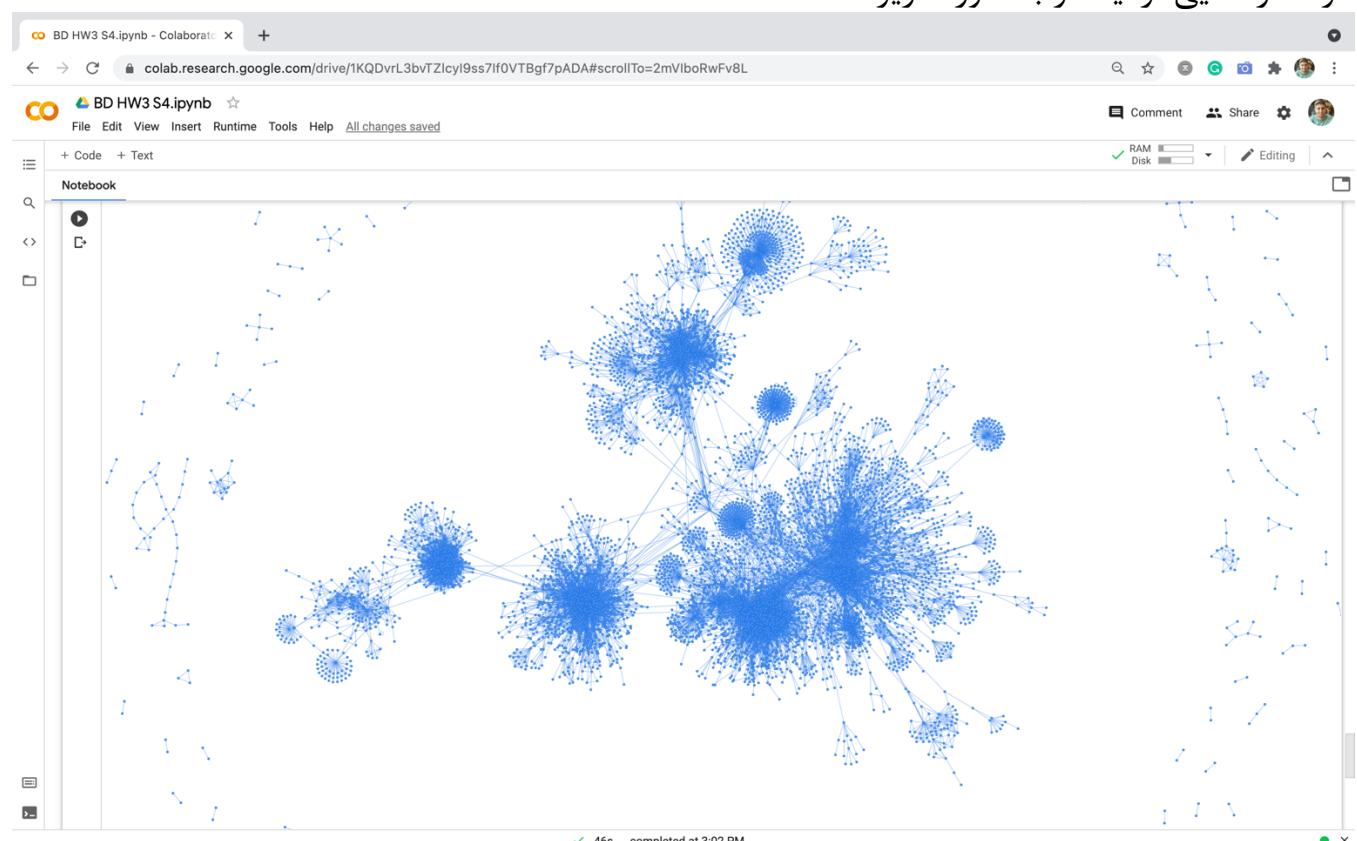


```
[ ] import networkx as nx
wiki = nx.from_pandas_edgelist(wiki_graph.edges.toPandas(), source='src', target='dst')

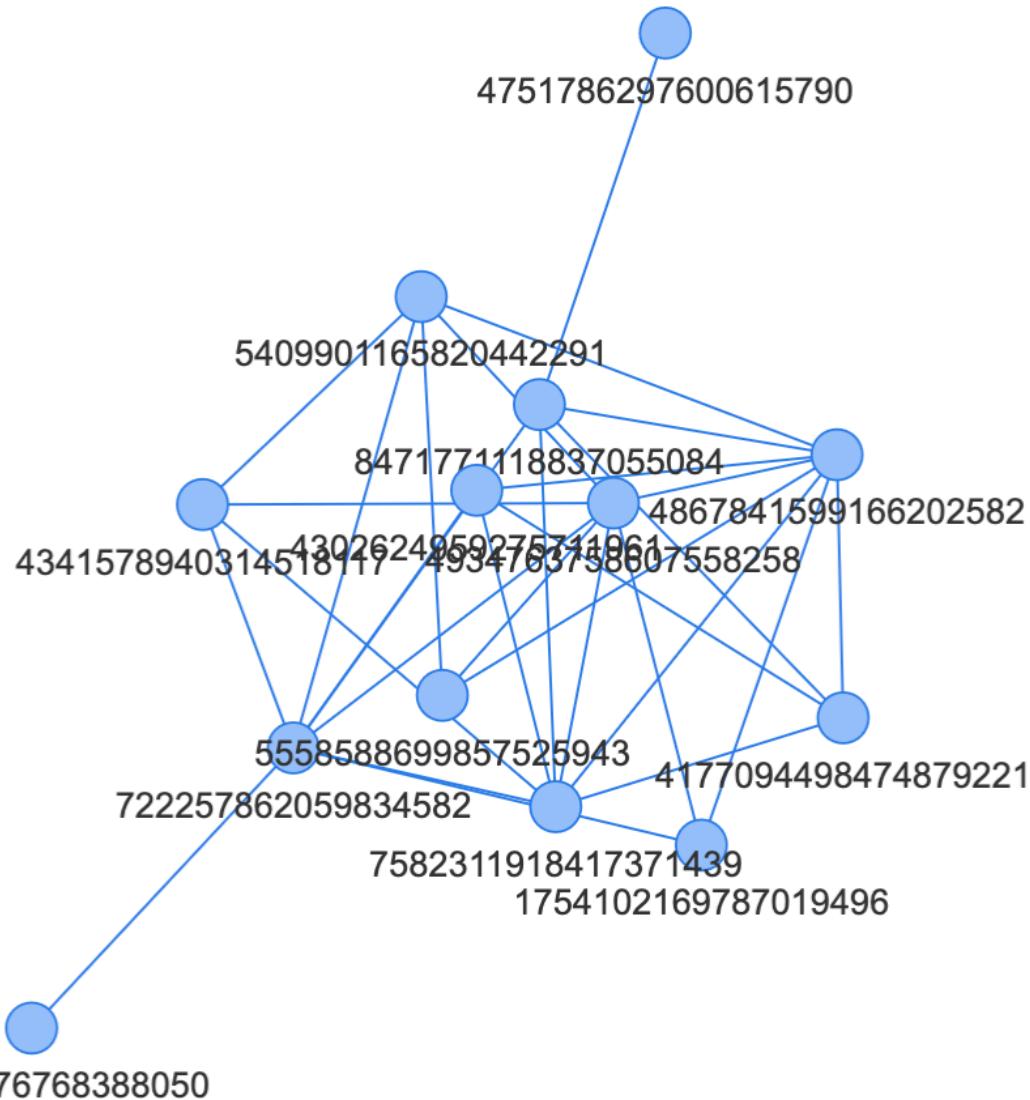
[ ] from dimcli.utils.networkviz import NetworkViz
viznet = NetworkViz(notebook=True, width="100%", height="800px")
viznet.from_nx(wiki)
viznet.show('wiki.html')
```



گراف از نمایی نزدیک‌تر به صورت زیر است.



خوشای از گراف از نمایی بسیار نزدیک نمایشگر شناسه‌ها نیز هست.



خود کتابخانه `networkx` نیز با استفاده از تابع `draw` گراف رسم می‌کند که برای گراف شلوغی مانند این خوب نیست و صرفاً یک تصویر است. بر خلاف `networkviz` که `interactive` است. خروجی `draw` مربوط به `networkx` به صورت زیر است.

