МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ
КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

ЗВІТ

про виконання лабораторної роботи №5
з дисципліни «Високопродуктивні розподілені системи»
на тему: «Spark»

Виконали:
студенти 5 курсу
групи КІ-31мп
Бірук С.,
Зиблий Є.,
Коваль М.,
Шатіхін Є.,
Перевірив:
Кухарєв С.М,

Київ – 2023

Передумови

1 Поставити та налаштувати Spark
2 На основі координат адрес Лондона наданих в даних:
(https://drive.google.com/drive/folders/1jtnV2uqNDg5JJE3h3CxJrE7KCrPyJiX4?usp=sharing)
згенерувати 10ГБ+ даних, які б описували поїздки таксі та характеризувалися :

- водієм
- клієнтом
- стартова точка
- кінцева точка
- час початку поїздки
- час кінця поїздки
- вартість поїздки(запропонуйте мені формулу, яка б враховувала час пік та
нічний час)
- оцінка водія (опціонально)
- відгук на водія (за категоріями) - опціонально
- текстовий відкук на водія (опціонально)
- оцінка на клієнта (опціонально)
- відгук на клієнта(за категоріями) - опціонально

Увага! ОПЦІОНАЛЬНО = може бути присутнім не у всіх поїздках (але в частині таки
має бути))))
водіїв 2к+
Клієнтів 4к+

3 на основі біг дата бази, що ви створили в п 2 отримати наступні “звіти” (використовуйте map reduce) та перевірити перевірки, що у вас потрібна кількість
клієнтів, водіїв, оцінок і тд (ви продумуєте які перевірки покажуть релевантність
генерації ваших даних)). Продумайте, що робити з текстовими коментами ? Враховуйте, що я можу попросити щось з ними зробити прямо на здачі лаби ;))

Лістинг програми

```
from pyspark.sql import SparkSession, functions
from pyspark.sql.functions import lit, col, rand, when
from math import sin, cos, sqrt, atan2
from datetime import datetime, timedelta
import numpy as np
import pandas as pd
import math
import random
from functools import reduce

def generate_normal_distribution(mean, std_dev):
    u = 1 - random.random()  # Converting [0,1) to (0,1]
    v = random.random()
    z = math.sqrt(-2.0 * math.log(u)) * math.cos(2.0 * math.pi * v)
    return z * std_dev + mean

driver_positive_feedbacks = [
    "Fast ride",
    "Polite driver",
    "Clean car",
    "Comfortable car",
    "Excellent service",
    "Great communication",
    "Prompt arrival",
]

driver_negative_feedbacks = [
    "Late arrival",
    "Rude driver",
    "Dirty car",
    "Uncomfortable ride",
    "Poor service",
    "Unsafe driving",
    "Ignored instructions",
    "Overcharged",
]

passenger_positive_feedbacks = [
    "Polite & friendly",
    "Pleasant conversation",
    "Clear instructions",
    "In time for pickup",
    "Clear car",
]

passenger_negative_feedbacks = [
    "Late for pickup",
    "Impolite and unfriendly behavior",
    "Messy and dirty in the car",
    "Didn't follow safety guidelines",
```

```
    "Provided unclear destination instructions",
    "Disruptive during the ride",
]

def generate_random_rating(skew):
    base_rating = random.random() * skew
    rand = generate_normal_distribution(3, 2)
    skewed_rating = abs(rand + base_rating)  # Calculate the skewed rating
    return min(5, skewed_rating)  # Ensure the rating is not greater than 5

def generate_feedback(is_driver, feedback_probability, skew=2):
    rating = generate_random_rating(skew)



    if 4.5 <= rating:
        num_feedbacks = 3
    elif 3.5 <= rating < 4.5:
        num_feedbacks = 2
    elif 2.5 <= rating < 3.5:
        num_feedbacks = 1
    elif 1.5 <= rating < 2.5:
        num_feedbacks = 2
    else:
        num_feedbacks = 3

    feedback_list = (
        driver_positive_feedbacks if is_driver and rating > 3
        else passenger_positive_feedbacks if not is_driver and rating > 3
        else driver_negative_feedbacks if is_driver
        else passenger_negative_feedbacks
    )

    selected_feedbacks = random.sample(feedback_list, num_feedbacks)

    if random.random() < feedback_probability:
        return {
            "Rating": np.round(rating, 0),
            "Notes": selected_feedbacks
        }
    else:
        return {
            "Rating": None,
            "Notes": []
        }

def calculate_distance(lat1, lon1, lat2, lon2):
    # Haversine formula to calculate distance between two points
    R = 6371  # Radius of Earth in kilometers
    d_lat = (lat2 - lat1) * (3.14159 / 180)
    d_lon = (lon2 - lon1) * (3.14159 / 180)
    a = (
        pow(sin(d_lat / 2), 2)
```

```
        + cos(lat1 * (3.14159 / 180)) * cos(lat2 * (3.14159 / 180)) * pow(sin(d_lon
/ 2), 2)
    )
    c = 2 * atan2(sqrt(a), sqrt(1 - a))
    distance = R * c  # Distance in kilometers
    return distance

def calculate_time_to_reach_destination(distance, speed):
    # Calculate time (in hours) to reach the destination at the given speed
    time_in_hours = distance / speed
    return time_in_hours * 60 * 60 * 1000  # Convert hours to milliseconds

def generate_route(start_date_time, start_lat, start_lon, end_lat, end_lon,
interval_seconds, speed):
    route = []

    end_date_time = start_date_time + timedelta(
        milliseconds=calculate_time_to_reach_destination(
            calculate_distance(start_lat, start_lon, end_lat, end_lon), speed
        )
    )

    route.append({
        "Location": {
            "type": "Point",
            "coordinates": [start_lon, start_lat],
        },
        "DateTime": start_date_time,
    })

    route.append({
        "Location": {
            "type": "Point",
            "coordinates": [end_lon, end_lat],
        },
        "DateTime": end_date_time,
    })

    return route

# Initialize Spark session
spark = SparkSession.builder.appName("LondonTaxiSimulation").getOrCreate()

postcodes = spark.read.option('header','true').csv('/content/drive/MyDrive/Colab
Notebooks/London postcodes.csv')
# postcodes =
((postcodes.unionAll(postcodes)).unionAll(postcodes)).unionAll(postcodes)
# print(pos tcodes.count())
postcodes.cache()

text_comments =
spark.read.option('header','true').csv('/content/drive/MyDrive/Colab
Notebooks/Uber_Ride_Reviews.csv')
```

```
text_comments.cache()

text_reviews_1 = text_comments.filter(text_comments.ride_rating == 1.0).collect()
text_reviews_2 = text_comments.filter(text_comments.ride_rating == 2.0).collect()
text_reviews_3 = text_comments.filter(text_comments.ride_rating == 3.0).collect()
text_reviews_4 = text_comments.filter(text_comments.ride_rating == 4.0).collect()
text_reviews_5 = text_comments.filter(text_comments.ride_rating == 5.0).collect()


# df.select("Postcode", "Latitude", "Longitude", "District").show()

# spark.stop()

debug = False

orders_to_create = 100000

drivers = 3000
driverFeedbackProb = 0.7
driverFeedbackCommentProb = 0.1

passengers = 5000
passengerFeedbackProb = 0.3

discreteTimeSeconds = 5
taxiAvgSpeed = 30
minDistance = 0.5

meanHour = 18
stdDevHour = 3


def generateRandomDateTime(meanHour, stdDevHour):
    randomHour = generate_normal_distribution(meanHour % 24, stdDevHour) % 24
    randomMinutes = random.randint(0, 59)
    randomSeconds = random.randint(0, 59)

    dateTime = datetime.now()
    dateTime = dateTime.replace(
        hour=int(randomHour), minute=randomMinutes, second=randomSeconds
    )

    return dateTime



def create_orders():
    orders = []
    read_rows = postcodes.orderBy(rand()).limit(2 * orders_to_create)
    read_rows.cache()
    read_rows_list = read_rows.collect()
    for j in range(orders_to_create):
    # for j in range(2):
```

```
        start_point = read_rows_list[j]
        # print("j + orders_to_create: ", j + orders_to_create)
        end_point = read_rows_list[j + orders_to_create]
        distance = calculate_distance(
        float(start_point["Latitude"]),
        float(start_point["Longitude"]),
        float(end_point["Latitude"]),
        float(end_point["Longitude"])
        )

        # if distance is less than permitted, resample
        while distance < minDistance:
            index = random.randint(0, int(read_rows.count()) - 1)
            while index == j:
                index = random.randint(0, int(read_rows.count()) - 1)
            end_point = read_rows_list[index]

            distance = calculate_distance(
            float(start_point["Latitude"]),
            float(start_point["Longitude"]),
            float(end_point["Latitude"]),
            float(end_point["Longitude"])
            )
            if debug:
                print(f"Resampled j={j} to index={index}, distance={distance}")

        # get driver
        driverId = random.randint(0, drivers - 1)
        driverFeedback = generate_feedback(True, driverFeedbackProb)
        # get passenger
        passengerId = random.randint(0, passengers - 1)
        passengerFeedback = generate_feedback(False, passengerFeedbackProb)
        # get time of ride
        time = generateRandomDateTime(meanHour, stdDevHour)
        # get route
        route = generate_route(
            time,
            float(start_point["Latitude"]),
            float(start_point["Longitude"]),
            float(end_point["Latitude"]),
            float(end_point["Longitude"]),
            discreteTimeSeconds,
            taxiAvgSpeed
        )

        driver_rating = driverFeedback["Rating"]

        if (random.random() < driverFeedbackCommentProb and (len(text_reviews_1) >
0 or len(text_reviews_2) > 0 or len(text_reviews_3) > 0 or len(text_reviews_4) > 0
or len(text_reviews_5) > 0)):
            if driver_rating == 1.0:
                driverFeedbackComment =
random.choice(text_reviews_1)["ride_review"]
```

```
            elif driver_rating == 2.0:
                driverFeedbackComment =
random.choice(text_reviews_2)["ride_review"]
            elif driver_rating == 3.0:
                driverFeedbackComment =
random.choice(text_reviews_3)["ride_review"]
            elif driver_rating == 4.0:
                driverFeedbackComment =
random.choice(text_reviews_4)["ride_review"]
            elif driver_rating == 5.0:
                driverFeedbackComment =
random.choice(text_reviews_5)["ride_review"]
            else:  driverFeedbackComment = ''
        else: driverFeedbackComment = ''

        # get duration
        duration = (route[-1]["DateTime"] - route[0]["DateTime"])  # in seconds
        # get price
        if (int(route[0]["DateTime"].hour) >= meanHour - stdDevHour &
int(route[0]["DateTime"].hour) < meanHour + stdDevHour):
            price = duration.total_seconds() * 0.03
        else:
            price = duration.total_seconds() * 0.02

        # compose row
        result = [
            driverId,
            driverFeedback["Rating"],
            driverFeedback["Notes"],
            driverFeedbackComment,
            passengerId,
            passengerFeedback["Rating"],
            passengerFeedback["Notes"],
            [start_point["Longitude"], start_point["Latitude"]],
            route[0]["DateTime"],
            [end_point["Longitude"], end_point["Latitude"]],
            route[-1]["DateTime"],
            distance,
            duration,
            np.round(float(price), 2)
        ]
        orders.append(result)
    read_rows.unpersist()
    return orders

# try:
print("Generating the data...")
orders = create_orders()

# print(orders)
# print("debug 1")
headers = [
        'DriverId',
```

```
        'DriverFeedbackRating',
        'DriverFeedbackNotes',
        'DriverFeedbackComment',
        'PassengerId',
        'PassengerFeedbackRating',
        'PassengerFeedbackNotes',
        'DepartureLocation',
        'DepartureTimestamp',
        'DestinationLocation',
        'DestinationTimestamp',
        'Distance',
        'Duration',
        'Price'
            ]

df = pd.DataFrame(data=orders,columns=headers)
orders_df = spark.createDataFrame(df)

print("The data is generated and put into a dataframe!")

    # result_df.repartition(1).write.mode('overwrite').csv("./output/",
header='True')
    #
result_df.coalesce(1).write.format("csv").option("header",True).option("sep","|").s
ave("./output/")

# except Exception as e:
#     print(f"An error occurred: {e}")

# finally:
#     print('End of session')
#     spark.stop()

######################## VARIANT 3 ########################

hour = functions.udf(lambda x: int(x.hour))
report1_df = orders_df.withColumn("DepartureHour", hour("DepartureTimestamp"))
# truncate=False

num_of_drivers =
orders_df.agg(functions.countDistinct("DriverId")).withColumnRenamed('count(DISTINC
T DriverId)', 'Number of drivers')
num_of_drivers.cache()
num_of_drivers.show()

num_of_passengers =
orders_df.agg(functions.countDistinct("PassengerId")).withColumnRenamed('count(DIST
INCT PassengerId)', 'Number of passengers')
num_of_passengers.cache()
num_of_passengers.show()

report1_df = report1_df.groupBy("DepartureHour").count().withColumnRenamed('count',
'CountByDepartureHour')
```

```
# report1_df.show()
report1_df =
report1_df.orderBy(col("CountByDepartureHour").desc()).limit(1).withColumnRenamed('
DepartureHour', 'Most popular departure hour')
report1_df.cache()
report1_df.show()

######################## VARIANT 1 ########################

num_of_drivers =
orders_df.agg(functions.countDistinct("DriverId")).withColumnRenamed('count(DISTINC
T DriverId)', 'Number of drivers')
num_of_drivers.cache()
num_of_drivers.show()

num_of_passengers =
orders_df.agg(functions.countDistinct("PassengerId")).withColumnRenamed('count(DIST
INCT PassengerId)', 'Number of passengers')
num_of_passengers.cache()
num_of_passengers.show()

report2_df =
orders_df.dropna().groupBy("DriverId").mean("DriverFeedbackRating").select("*",func
tions.round("avg(DriverFeedbackRating)",2)).withColumnRenamed('round(avg(DriverFeed
backRating), 2)', 'Average driver rating').drop('avg(DriverFeedbackRating)')
# report1_df.show()
report2_df = report2_df.orderBy(col("Average driver
rating").desc()).limit(100).withColumnRenamed('DriverId', 'Top 100 best rated
drivers')
report2_df.cache()
report2_df.show(100)

orders_df.show(50)

text_comments.show(10,False)
```

Результати роботи програми

Приклад згенерованого датасету

DriverId	DriverFeedbackRating	DriverFeedbackNotes	DriverFeedbackComment	PassengerId	PassengerFeedbackRating	PassengerFeedbackNotes
391	2.0	[Late arrival, Un...		1941	NaN	[]
2665	1.0	[Poor service, Ru...	They charging unf...	1868	NaN	[]
1393	NaN	[]		3038	2.0	[Messy and dirty ...
976	NaN	[]		1537	NaN	[]
2952	5.0	[Excellent servic...		4771	2.0	[Late for pickup,...
2148	NaN	[]		1365	NaN	[]
573	5.0	[Prompt arrival, ...		4624	5.0	[Polite & friendl...
242	NaN	[]		778	NaN	[]
1298	2.0	[Uncomfortable ri...		84	NaN	[]
1429	NaN	[]		4753	5.0	[Clear instructio...
2752	NaN	[]		4550	NaN	[]
2004	NaN	[]		4304	NaN	[]
968	NaN	[]		4555	5.0	[Polite & friendl...
1535	2.0	[Overcharged, Lat...		4941	4.0	[Pleasant convers...
456	NaN	[]		42	NaN	[]
1972	4.0	[Polite driver, C...		3364	NaN	[]
874	NaN	[]		163	NaN	[]
666	NaN	[]		2780	NaN	[]
2586	5.0	[Clean car, Excel...		4352	NaN	[]
675	4.0	[Prompt arrival, ...	My first experien...	4749	NaN	[]
2876	5.0	[Prompt arrival, ...		2490	NaN	[]
2858	5.0	[Clean car, Polit...		4561	NaN	[]
1122	0.0	[Late arrival, Di...		3029	NaN	[]
2326	NaN	[]		3135	NaN	[]
2209	2.0	[Late arrival, Ov...		3166	3.0	[Didn't follow sa...
294	5.0	[Clean car, Great...		4204	2.0	[Impolite and unf...
2392	2.0	[Ignored instruct...		1446	5.0	[In time for pick...
2628	NaN	[]		933	NaN	[]
1296	5.0	[Fast ride, Clean...		703	NaN	[]
242	3.0	[Ignored instruct...		2556	NaN	[]

DepartureLocation	DepartureTimestamp	DestinationLocation	DestinationTimestamp	Distance	Duration	Price
[-0.058075, 51.51...	2023-12-20 16:01:...	[0.097087, 51.549...	2023-12-20 16:23:...	11.25908950137335	INTERVAL '0 00:22...	40.53
[0.111771, 51.38251]	2023-12-20 16:56:...	[-0.11632, 51.546...	2023-12-20 17:44:...	24.142246192574618	INTERVAL '0 00:48...	86.91
[-0.146528, 51.52...	2023-12-20 21:20:...	[-0.293398, 51.41...	2023-12-20 21:51:...	15.701542387314955	INTERVAL '0 00:31...	56.53
[-0.103543, 51.53...	2023-12-20 19:49:...	[-0.414849, 51.52...	2023-12-20 20:33:...	21.62457067427504	INTERVAL '0 00:43...	77.85
[-0.13543, 51.530...	2023-12-20 15:24:...	[0.19212, 51.516765]	2023-12-20 16:09:...	22.714877819968503	INTERVAL '0 00:45...	81.77
[-0.180728, 51.61...	2023-12-20 18:31:...	[0.062789, 51.523...	2023-12-20 19:11:...	19.772895898748875	INTERVAL '0 00:39...	71.18
[-0.029651, 51.53...	2023-12-20 18:16:...	[-0.050576, 51.48...	2023-12-20 18:26:...	4.992447454722651	INTERVAL '0 00:09...	17.97
[0.033902, 51.529...	2023-12-20 19:43:...	[-0.099435, 51.37...	2023-12-20 20:22:...	19.392587346601605	INTERVAL '0 00:38...	69.81
[0.038183, 51.483...	2023-12-20 09:49:...	[-0.131484, 51.36...	2023-12-20 10:25:...	17.783841459982487	INTERVAL '0 00:35...	64.02
[-0.190105, 51.38...	2023-12-20 13:44:...	[-0.393965, 51.57...	2023-12-20 14:35:...	25.445932234460663	INTERVAL '0 00:50...	91.61
[-0.140345, 51.54...	2023-12-20 20:14:...	[-0.065687, 51.45...	2023-12-20 20:36:...	11.285556999366335	INTERVAL '0 00:22...	40.63
[0.123553, 51.497...	2023-12-20 15:36:...	[-0.370053, 51.41...	2023-12-20 16:47:...	35.413706308767146	INTERVAL '0 01:10...	127.49
[-0.099129, 51.48...	2023-12-20 16:39:...	[-0.303012, 51.41...	2023-12-20 17:11:...	15.942610853190688	INTERVAL '0 00:31...	57.39
[-0.112088, 51.52...	2023-12-20 22:11:...	[-0.339032, 51.57...	2023-12-20 22:45:...	16.794147324919795	INTERVAL '0 00:33...	60.46
[-0.335118, 51.46...	2023-12-20 17:56:...	[0.031264, 51.622...	2023-12-20 18:58:...	31.084321798990445	INTERVAL '0 01:02...	111.9
[-0.112042, 51.52...	2023-12-20 13:38:...	[-0.102062, 51.51...	2023-12-20 13:41:...	1.02446268578059	INTERVAL '0 00:02...	3.69
[0.026654, 51.619...	2023-12-20 16:32:...	[-0.028302, 51.51...	2023-12-20 16:57:...	12.434429459195805	INTERVAL '0 00:24...	44.76
[-0.011132, 51.47...	2023-12-20 19:35:...	[-0.317013, 51.43...	2023-12-20 20:19:...	21.67380020888414	INTERVAL '0 00:43...	78.03
[-0.154551, 51.51...	2023-12-20 19:53:...	[-0.13543, 51.530...	2023-12-20 19:57:...	1.8946302319912804	INTERVAL '0 00:03...	6.82
[-0.29443, 51.463...	2023-12-20 16:45:...	[-0.355862, 51.58...	2023-12-20 17:13:...	14.16958962949949	INTERVAL '0 00:28...	51.01
[-0.209349, 51.52...	2023-12-20 14:37:...	[-0.112847, 51.51...	2023-12-20 14:51:...	6.8141136743085156	INTERVAL '0 00:13...	24.53
[-0.17332, 51.532...	2023-12-20 21:19:...	[-0.199134, 51.52...	2023-12-20 21:23:...	2.0239451223115776	INTERVAL '0 00:04...	7.29
[0.007223, 51.412...	2023-12-20 12:03:...	[-0.138822, 51.51...	2023-12-20 12:34:...	15.435180038023622	INTERVAL '0 00:30...	55.57
[0.074663, 51.596...	2023-12-20 16:10:...	[-0.306002, 51.41...	2023-12-20 17:16:...	33.30245915474044	INTERVAL '0 01:06...	119.89
[-0.171953, 51.35...	2023-12-20 19:30:...	[-0.20621, 51.495...	2023-12-20 20:02:...	16.37098449947839	INTERVAL '0 00:32...	58.94
[-0.055578, 51.43...	2023-12-20 16:53:...	[-0.29759, 51.410...	2023-12-20 17:27:...	16.99936642177357	INTERVAL '0 00:33...	61.2
[-0.134608, 51.54...	2023-12-20 17:38:...	[-0.091863, 51.37...	2023-12-20 18:16:...	19.083370970963074	INTERVAL '0 00:38...	68.7
[-0.100636, 51.52...	2023-12-20 16:48:...	[-0.19773, 51.535...	2023-12-20 17:02:...	6.756664798902437	INTERVAL '0 00:13...	24.32
[-0.072414, 51.46...	2023-12-20 16:59:...	[-0.237903, 51.41...	2023-12-20 17:26:...	13.09359813789207	INTERVAL '0 00:26...	47.14
[-0.079272, 51.37...	2023-12-20 16:18:...	[-0.298807, 51.46...	2023-12-20 16:54:...	17.78561903522927	INTERVAL '0 00:35...	64.03

Варіант 3

- в який проміжок часу здійснюється найбільше поїздок

```
+-----------------+
|Number of drivers|
+-----------------+
|             3000|
+-----------------+

+--------------------+
|Number of passengers|
+--------------------+
|                5000|
+--------------------+

+---------------------------+--------------------+
|Most popular departure hour|CountByDepartureHour|
+---------------------------+--------------------+
|                         18|               13263|
+---------------------------+--------------------+
```

Варіант 1

- топ 100 водіїв за рейтингом

Number of drivers
3000

Number of passengers
5000

Top 100 best rated drivers	Average driver rating
2906	5.0
1905	5.0
149	5.0
110	5.0
430	5.0
1246	5.0
1773	5.0
389	5.0
900	5.0
1136	5.0
328	5.0
903	5.0
2017	5.0
2070	5.0
2547	5.0
1007	5.0
1340	5.0
2466	5.0
491	5.0
2757	5.0
2945	5.0
2569	5.0
2754	5.0
1753	5.0
916	5.0
2230	5.0
2258	5.0
913	5.0
1226	5.0
643	5.0
2042	5.0
70	5.0
1004	5.0
2645	5.0
1564	5.0
633	5.0
2375	5.0
2542	5.0
456	5.0
193	5.0
2444	5.0
2656	5.0
2103	5.0
1457	5.0
1492	5.0
2934	4.88
1627	4.86
1785	4.83
975	4.83
1034	4.83
1303	4.83
2776	4.83
2175	4.8
964	4.8
2223	4.8
834	4.8
2752	4.8
134	4.8
2150	4.8
547	4.8
209	4.78
2536	4.78
2657	4.78
1302	4.75
1219	4.75
1441	4.75
1721	4.75