# BIG DATA

Student: Teodora Kocić
Profesor: Dragan Stojanović

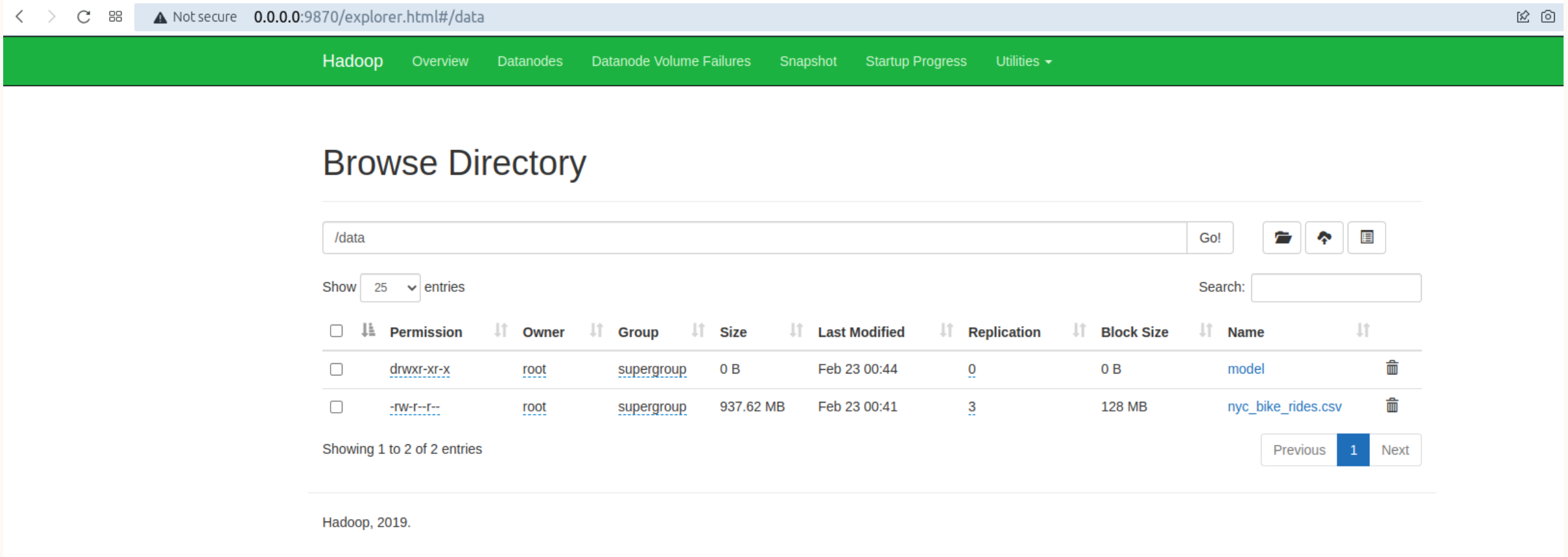
# SADRŽAJ

# KORIŠĆENI PODACI

- Dataset koji je iskorišćen je **Bike New York City** (podaci o vožnjama iz 2013. godine u periodu od jula zaključno sa mesecom decembrom)
- Dataset sadrži podatke o vremenu trajanja svake zabeležene vožnje, datumu i tačnom vremenu početka, odnosno kraja vožnje, stanici sa koje je biciklista krenuo i stanici na kojoj je završio datu vožnju, dodatno dataset nosi podatke i o serijskom broju bicikla i jedinstvenim brojevima stanica (IDs), lokaciji stanica (latituda i longituda), tipu, polu i godini rođenja bicikliste

# DATASET NA HDFS-U





# OPIS PROJEKTA 1

- Koriste se različite funkcije agregacije i funkcije za rad sa okvirima podataka (DataFrame) iz Pyspark biblioteke
- Filtriranje podataka i sortiranje dobijenih rezultata
- Tabelarni prikaz rezultata i snimanje pojedinačnih rezultata u fajl
- Lokalno izvršenje i izvršenje na klasteru

# IMPLEMENTACIJA PROJEKTA 1

Značajni delovi

# PYSPARK

## LOKALNO IZVRŠENJE

- 6 različitih job-ova
- Incijalni dataset se redukuje tako da sadrži redove koji zadovoljavaju uslove da je početna stanica severno od Kipa slobode, a krajnja stanica zapadno od aerodroma John F. Kennedy
- Vremensko ograničenje je da se izdvajaju redovi kod kojih je početak vožnje bio bilo kog dana nakon 15. avgusta i to nakon 6 h prepodne
- Dodatno i izračunavanja pređenog puta i srednje brzine (numpy biblioteka)

## IZVRŠENJE NA KLASTERU

- 6 različitih job-ova
- Incijalni dataset se redukuje tako da sadrži redove koji zadovoljavaju uslove da je početna stanica severno od Kipa slobode, a krajnja stanica zapadno od aerodroma John F. Kennedy
- Vremensko ograničenje je da se izdvajaju redovi kod kojih je početak vožnje bio bilo kog dana nakon 15. avgusta i to nakon 6 h prepodne
- Praćenje izvršenja ovih poslova po fazama na spark master-u i dva spark radnika

# TASKOVI



**TASK 1**

Prikazuje prvih 50 redova filtriranih podataka

**TASK 2**

Pronalazi rutu vožnje kod koje je standardna devijacija vremena vožnje bila najmanja

**TASK 3**

Za oba tipa korisnika se određuje prosečno vreme vožnje

**TASK 4**

Određivanje broja korisnika rođenih nakon 1990. godine, a sortiranih po dužini prosečnog vremena trajanja vožnje

**TASK 5**

Broj biciklista koji je vožnju završio na nekoj od Avenija, sortiranih po srednjem vremenu trajanja vožnje

**TASK 6**

Od ženskih biciklista pronalazi se ona koja je u vožnji provela najduži period, mereno u časovima

# ZNAČAJNI DELOVI KODA



```
END_LOCATION = float(os.getenv('END_LOCATION'))
TIME_START = int(os.getenv('TIME_START'))
FILE_DATA = os.getenv('FILE_DATA')
HDFS_DATA = os.getenv('HDFS_DATA')

lines = list()

# spark = SparkSession.builder.master('local[*]').appName('Big Data 1').getOrCreate()
# tStart = datetime.now()
spark = SparkSession.builder.appName('Big Data 1').getOrCreate()

rideSchema = StructType().add('tripduration', 'integer').add('starttime', 'string').add('stoptime', 'string').add('start_station_id', 'integer')\
                         .add('start_station_name', 'string').add('start_station_latitude', 'float').add('start_station_longitude', 'float')\
                         .add('end_station_id', 'integer').add('end_station_name', 'string').add('end_station_latitude', 'float')\
                         .add('end_station_longitude', 'float').add('bikeid', 'integer').add('usertype', 'string')\
                         .add('birthyear', 'string').add('gender', 'integer')

# dataFrame = spark.read.csv(FILE_DATA, schema=rideSchema)       # For local execution
dataFrame = spark.read.csv(HDFS_DATA, schema=rideSchema)

datetimeArg = datetime(2013, 8, 15, 00, TIME_START, 00)

dataFrame = dataFrame.withColumn('starttimeFormat', to_timestamp(dataFrame.starttime, 'yyyy-MM-dd HH:mm:ss'))
dataFrame = dataFrame.withColumn('start_location', array(array(dataFrame.start_station_latitude, dataFrame.start_station_longitude)))
dataFrame = dataFrame.withColumn('end_location', array(array(dataFrame.end_station_latitude, dataFrame.end_station_longitude)))

# This part for spark-local
# dataFrame = dataFrame.withColumn('distance', geodesic_udf(col('start_location'), col('end_location')))
# dataFrame = dataFrame.withColumn('Vsr', col('distance')/col('tripduration'))
# dataFrame = dataFrame.withColumn('Vsr', col('Vsr').cast('float'))
# dataFrame_filtered = dataFrame_filtered.select('start_station_name', 'end_station_name', 'distance', 'tripduration', 'Vsr', 'usertype', 'birthyear', 'gender')
# dataFrame_filtered_mean_time = dataFrame_filtered.groupBy('birthyear').agg(avg('Vsr').alias('average_Vsr'), mean('tripduration').alias('mean_trip_time')).show(truncate=False)
# dataFrame_filtered_trip_data = dataFrame_filtered.groupBy('usertype').agg(max('tripduration').alias('max_trip_duration'), min('tripduration').alias('min_trip_duration'), sum('distance

dataFrame_filtered = dataFrame.filter((dataFrame.start_station_latitude > START_LOCATION) & (dataFrame.end_station_longitude < END_LOCATION) & \
    (dataFrame.starttimeFormat > datetimeArg))

task1 = 'First 50 rows of filtered data are:'
print(task1, '\n')
dataFrame_filtered.show(n=50, truncate=False)

dataFrame_filtered_location_start_stop = dataFrame_filtered.groupBy('start_station_name', 'end_station_name').agg(stddev('tripduration')\
    .alias('standard_deviation_time_spent_between_stations')).sort(col('standard_deviation_time_spent_between_stations').asc()).collect()
task2 = 'The lowest standard deviation for the trip duration is for the route ' + str(dataFrame_filtered_location_start_stop[0].asDict()['start_station_name']) \
    + '-' + str(dataFrame_filtered_location_start_stop[0].asDict()['end_station_name']) + '.'
print(task2, '\n')
lines.append(task2)
```

Spark session

Read file from hdfs

Prepare dataset

Filter data

Task 1

Task 2

# ZNAČAJNI DELOVI KODA



```
dataFrame_filtered_longer_10_min = dataFrame_filtered.groupBy('usertype').agg(avg('tripduration').alias('average_trip_time')).filter(col('average_trip_time') > 600).collect()
task3 = str(dataFrame_filtered_longer_10_min[0].asDict()['usertype']) + 's have average trip duration time of ' + \
    str(dataFrame_filtered_longer_10_min[0].asDict()['average_trip_time'] / 60)+ ' minutes, while the same parametes is '\
        + str(dataFrame_filtered_longer_10_min[1].asDict()['average_trip_time'] / 60) + ' minutes for the ' + str(dataFrame_filtered_longer_10_min[1].asDict()['usertype']) + 's.'
print(task3, '\n')
lines.append(task3)

dataFrame_filtered_years = dataFrame_filtered.groupBy('birthyear').agg(count('*').alias('count'), avg('tripduration').alias('avg_riding_time'))\
    .filter(col('birthyear').startswith('199')).sort(col('avg_riding_time').desc()).collect()
for row in dataFrame_filtered_years:
    task4 = 'Number of riders who are ' + str(2013 - int(row.asDict()['birthyear'])) + ' years old is ' + str(row.asDict()['count']) \
        + ' and average riding time for those riders is ' + str(row.asDict()['avg_riding_time']) + ' seconds.'
    print(task4, '\n')
    lines.append(task4)

dataFrame_filtered_end_location = dataFrame_filtered.groupBy('end_station_name').agg(count('bikeid').alias('count_rides'), mean('tripduration')\
    .alias('mean_trip_time')).filter(col('end_station_name').like('%Avenue%')).sort(col('mean_trip_time').asc()).collect()
for row in dataFrame_filtered_end_location:
    task5 = 'Number of bike rides that ended on the Avenue \'' + str(row.asDict()['end_station_name']) + '\' is ' + str(row.asDict()['count_rides']) \
        + ' with mean time of ' + str(row.asDict()['mean_trip_time']) + ' seconds.'
    print(task5, '\n')
    lines.append(task5)

dataFrame_filtered_trip_data = dataFrame_filtered.groupBy('bikeId', 'gender').agg(max('tripduration').alias('max_trip_duration'), min('tripduration')\
    .alias('min_trip_duration'), sum('tripduration').alias('total_time'))\
    .filter(col('gender') == 2).sort(col('max_trip_duration').desc())
task6 = 'Woman who spent the most time driving, rode a bicycle overall ' + str(dataFrame_filtered_trip_data.collect()[0].asDict()['total_time'] / 3600) + ' hours.'
print(task6, '\n')
lines.append(task6)

with open("app/output.txt", "w") as fileOutput:
    for line in lines:
        fileOutput.write(line)
        fileOutput.write("\n")

# tEnd = datetime.now()
# delta = tEnd - tStart

# print('Spark execution was ' + str(delta) + ' long.')
```

Task 3

Task 4

Task 5

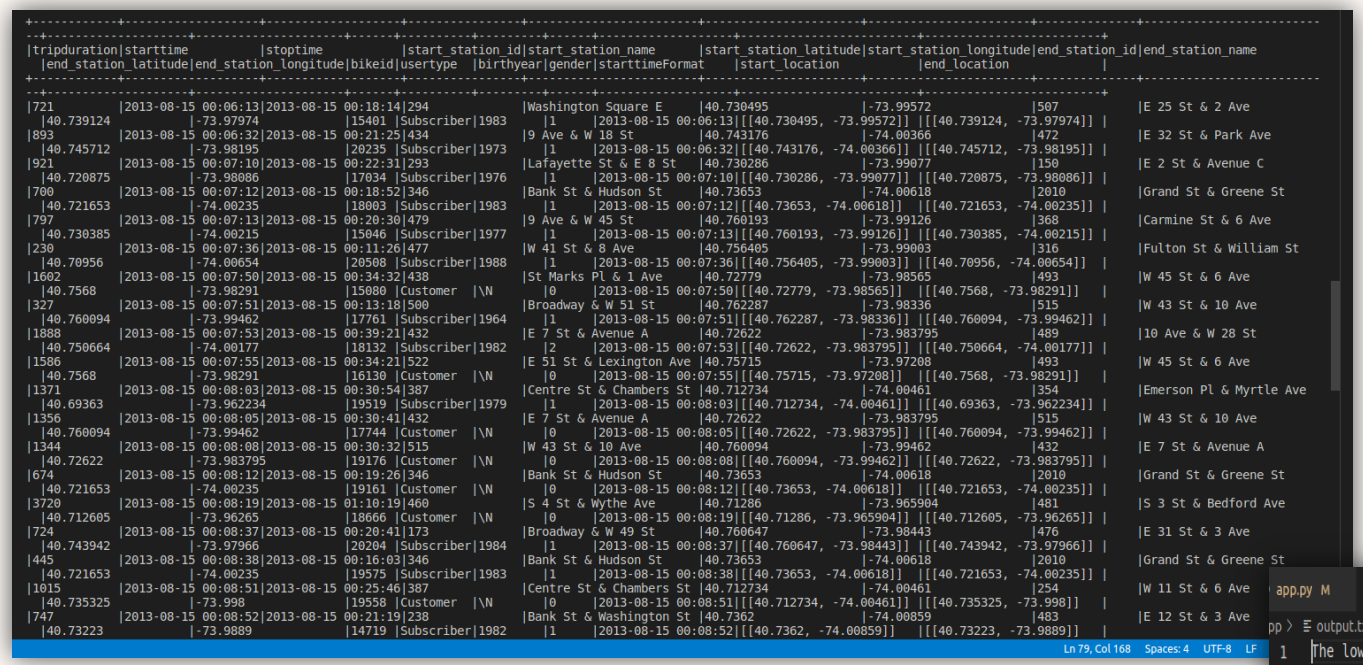
Task 6

Write result into txt file

# LOKALNO IZVRŠENJE I REZULTAT



TASK 1



| tripduration | starttime | stoptime | start_station_id | start_station_name | start_station_latitude | start_station_longitude | end_station_id | end_station_name | end_station_latitude | end_station_longitude | bikeid | usertype | birthyear | gender | starttimeFormat | start_location | end_location |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 721 | 2013-08-15 00:06:13 | 2013-08-15 00:18:14 | 294 | Washington Square E | 40.730495 | -73.99572 | 507 | E 25 St & 2 Ave | 40.739124 | -73.97974 | 15401 | Subscriber | 1983 | 1 | 2013-08-15 00:06:13 | [[40.730495, -73.99572]] | [[40.739124, -73.97974]] |
| 893 | 2013-08-15 00:06:32 | 2013-08-15 00:21:25 | 434 | 9 Ave & W 18 St | 40.743176 | -74.00366 | 472 | E 32 St & Park Ave | 40.745712 | -73.98195 | 20235 | Subscriber | 1973 | 1 | 2013-08-15 00:06:32 | [[40.743176, -74.00366]] | [[40.745712, -73.98195]] |
| 921 | 2013-08-15 00:07:10 | 2013-08-15 00:22:31 | 293 | Lafayette St & E 8 St | 40.730286 | -73.99077 | 150 | E 2 St & Avenue C | 40.720875 | -73.98086 | 17034 | Subscriber | 1976 | 1 | 2013-08-15 00:07:10 | [[40.730286, -73.99077]] | [[40.720875, -73.98086]] |
| 700 | 2013-08-15 00:07:12 | 2013-08-15 00:18:52 | 346 | Bank St & Hudson St | 40.73653 | -74.00618 | 2010 | Grand St & Greene St | 40.721653 | -74.00235 | 18003 | Subscriber | 1983 | 1 | 2013-08-15 00:07:12 | [[40.73653, -74.00618]] | [[40.721653, -74.00235]] |
| 797 | 2013-08-15 00:07:13 | 2013-08-15 00:20:30 | 479 | 9 Ave & W 45 St | 40.760193 | -73.99126 | 368 | Carmine St & 6 Ave | 40.730385 | -74.00215 | 15046 | Subscriber | 1977 | 1 | 2013-08-15 00:07:13 | [[40.760193, -73.99126]] | [[40.730385, -74.00215]] |
| 230 | 2013-08-15 00:07:36 | 2013-08-15 00:11:26 | 477 | W 41 St & 8 Ave | 40.756405 | -73.99003 | 316 | Fulton St & William St | 40.70956 | -74.00654 | 20508 | Subscriber | 1988 | 1 | 2013-08-15 00:07:36 | [[40.756405, -73.99003]] | [[40.70956, -74.00654]] |
| 1602 | 2013-08-15 00:07:50 | 2013-08-15 00:34:32 | 438 | St Marks Pl & 1 Ave | 40.72779 | -73.98565 | 493 | W 45 St & 6 Ave | 40.7568 | -73.98291 | 15080 | Customer | \N | 0 | 2013-08-15 00:07:50 | [[40.72779, -73.98565]] | [[40.7568, -73.98291]] |
| 327 | 2013-08-15 00:07:51 | 2013-08-15 00:13:18 | 500 | Broadway & W 51 St | 40.762287 | -73.98336 | 515 | W 43 St & 10 Ave | 40.760094 | -73.99462 | 17761 | Subscriber | 1964 | 1 | 2013-08-15 00:07:51 | [[40.762287, -73.98336]] | [[40.760094, -73.99462]] |
| 1888 | 2013-08-15 00:07:53 | 2013-08-15 00:39:21 | 432 | E 7 St & Avenue A | 40.72622 | -73.983795 | 489 | 10 Ave & W 28 St | 40.750664 | -74.00177 | 18132 | Subscriber | 1982 | 2 | 2013-08-15 00:07:53 | [[40.72622, -73.983795]] | [[40.750664, -74.00177]] |
| 1586 | 2013-08-15 00:07:55 | 2013-08-15 00:34:21 | 522 | E 51 St & Lexington Ave | 40.75715 | -73.97208 | 493 | W 45 St & 6 Ave | 40.7568 | -73.98291 | 16130 | Customer | \N | 0 | 2013-08-15 00:07:55 | [[40.75715, -73.97208]] | [[40.7568, -73.98291]] |
| 1371 | 2013-08-15 00:08:03 | 2013-08-15 00:30:54 | 387 | Centre St & Chambers St | 40.712734 | -74.00461 | 354 | Emerson Pl & Myrtle Ave | 40.69363 | -73.962234 | 19519 | Subscriber | 1979 | 1 | 2013-08-15 00:08:03 | [[40.712734, -74.00461]] | [[40.69363, -73.962234]] |
| 1356 | 2013-08-15 00:08:05 | 2013-08-15 00:30:41 | 432 | E 7 St & Avenue A | 40.72622 | -73.983795 | 515 | W 43 St & 10 Ave | 40.760094 | -73.99462 | 17744 | Customer | \N | 0 | 2013-08-15 00:08:05 | [[40.72622, -73.983795]] | [[40.760094, -73.99462]] |
| 1344 | 2013-08-15 00:08:08 | 2013-08-15 00:30:32 | 515 | W 43 St & 10 Ave | 40.760094 | -73.99462 | 432 | E 7 St & Avenue A | 40.72622 | -73.983795 | 19176 | Customer | \N | 0 | 2013-08-15 00:08:08 | [[40.760094, -73.99462]] | [[40.72622, -73.983795]] |
| 674 | 2013-08-15 00:08:12 | 2013-08-15 00:19:26 | 346 | Bank St & Hudson St | 40.73653 | -74.00618 | 2010 | Grand St & Greene St | 40.721653 | -74.00235 | 19161 | Customer | \N | 0 | 2013-08-15 00:08:12 | [[40.73653, -74.00618]] | [[40.721653, -74.00235]] |
| 3720 | 2013-08-15 00:08:19 | 2013-08-15 01:10:19 | 460 | S 4 St & Wythe Ave | 40.71286 | -73.965904 | 481 | S 3 St & Bedford Ave | 40.712605 | -73.96265 | 18666 | Customer | \N | 0 | 2013-08-15 00:08:19 | [[40.71286, -73.965904]] | [[40.712605, -73.96265]] |
| 724 | 2013-08-15 00:08:37 | 2013-08-15 00:20:41 | 173 | Broadway & W 49 St | 40.760647 | -73.98443 | 476 | E 31 St & 3 Ave | 40.743942 | -73.97966 | 20204 | Subscriber | 1984 | 1 | 2013-08-15 00:08:37 | [[40.760647, -73.98443]] | [[40.743942, -73.97966]] |
| 445 | 2013-08-15 00:08:38 | 2013-08-15 00:16:03 | 346 | Bank St & Hudson St | 40.73653 | -74.00618 | 2010 | Grand St & Greene St | 40.721653 | -74.00235 | 19575 | Subscriber | 1983 | 1 | 2013-08-15 00:08:38 | [[40.73653, -74.00618]] | [[40.721653, -74.00235]] |
| 1015 | 2013-08-15 00:08:51 | 2013-08-15 00:25:46 | 387 | Centre St & Chambers St | 40.712734 | -74.00461 | 254 | W 11 St & 6 Ave | 40.735325 | -73.998 | 19558 | Customer | \N | 0 | 2013-08-15 00:08:51 | [[40.712734, -74.00461]] | [[40.735325, -73.998]] |
| 747 | 2013-08-15 00:08:52 | 2013-08-15 00:21:19 | 238 | Bank St & Washington St | 40.7362 | -74.00859 | 483 | E 12 St & 3 Ave | 40.73223 | -73.9889 | 14719 | Subscriber | 1982 | 1 | 2013-08-15 00:08:52 | [[40.7362, -74.00859]] | [[40.73223, -73.9889]] |

TASK 2, 3, 4, 5, 6

app.py M  output.txt U

app > output.txt

```
The lowest standard deviation for the trip duration is for the route Broadway & Berry St-E 30 St & Park Ave S.
Subscribers have average trip duration time of 12.752130037959196 minutes, while the same parametes is 25.458552089991052 minutes for the Customers.
Number of riders who are 16 years old is 3053 and average riding time for those riders is 792.7605633802817 seconds.
Number of riders who are 17 years old is 4398 and average riding time for those riders is 781.8703956343793 seconds.
Number of riders who are 18 years old is 8004 and average riding time for those riders is 746.9172913543229 seconds.
Number of riders who are 22 years old is 40451 and average riding time for those riders is 741.2251860275395 seconds.
Number of riders who are 23 years old is 73121 and average riding time for those riders is 727.6273710698705 seconds.
Number of riders who are 21 years old is 29886 and average riding time for those riders is 639.4023288496286 seconds.
Number of riders who are 19 years old is 13285 and average riding time for those riders is 622.0160331200602 seconds.
Number of riders who are 20 years old is 23491 and average riding time for those riders is 616.4971265591078 seconds.
Number of bike rides that ended on the Avenue 'E 13 St & Avenue A' is 15074 with mean time of 773.0951306886029 seconds.
Number of bike rides that ended on the Avenue 'E 9 St & Avenue C' is 11409 with mean time of 774.7850819528443 seconds.
Number of bike rides that ended on the Avenue 'E 2 St & Avenue B' is 15042 with mean time of 801.3648451003855 seconds.
Number of bike rides that ended on the Avenue 'E 7 St & Avenue A' is 12676 with mean time of 803.0022877879458 seconds.
Number of bike rides that ended on the Avenue 'E 5 St & Avenue C' is 6645 with mean time of 807.2484574868322 seconds.
Number of bike rides that ended on the Avenue 'E 6 St & Avenue B' is 13241 with mean time of 809.2239256853712 seconds.
Number of bike rides that ended on the Avenue 'E 2 St & Avenue C' is 7560 with mean time of 855.6257936507936 seconds.
Number of bike rides that ended on the Avenue 'Avenue D & E 8 St' is 3434 with mean time of 891.8803145020385 seconds.
Number of bike rides that ended on the Avenue 'Avenue D & E 3 St' is 4240 with mean time of 903.0268867924528 seconds.
Number of bike rides that ended on the Avenue 'E 14 St & Avenue B' is 10496 with mean time of 917.922256097561 seconds.
Number of bike rides that ended on the Avenue 'E 6 St & Avenue D' is 4876 with mean time of 1030.3999179655455 seconds.
Number of bike rides that ended on the Avenue 'E 10 St & Avenue A' is 15715 with mean time of 1047.9067769646833 seconds.
Number of bike rides that ended on the Avenue 'Avenue D & E 12 St' is 1991 with mean time of 1376.95429432446 seconds.
Woman who spent the most time driving, rode a bicycle overall 587.0263888888888 hours.
```

# IZVRŠENJE NA KLASTERU



## Spark Master at spark://515c4525e254:7077

**URL:** spark://515c4525e254:7077
**Alive Workers:** 2
**Cores in use:** 16 Total, 16 Used
**Memory in use:** 28.8 GiB Total, 2.0 GiB Used
**Resources in use:**
**Applications:** 1 Running, 1 Completed
**Drivers:** 0 Running, 0 Completed
**Status:** ALIVE

### Workers (2)

| Worker Id | Address | State | Cores | Memory | Resources |
|---|---|---|---|---|---|
| worker-20230124184613-172.18.0.8-41479 | 172.18.0.8:41479 | ALIVE | 8 (8 Used) | 14.4 GiB (1024.0 MiB Used) | |
| worker-20230124184613-172.18.0.9-39357 | 172.18.0.9:39357 | ALIVE | 8 (8 Used) | 14.4 GiB (1024.0 MiB Used) | |

### Running Applications (1)

| Application ID | Name | Cores | Memory per Executor | Resources Per Executor | Submitted Time | User | State | Duration |
|---|---|---|---|---|---|---|---|---|
| app-20230124225510-0001 (kill) | Big Data 1 | 16 | 1024.0 MiB | | 2023/01/24 22:55:10 | root | RUNNING | 0.1 s |

### Completed Applications (1)

| Application ID | Name | Cores | Memory per Executor | Resources Per Executor | Submitted Time | User | State | Duration |
|---|---|---|---|---|---|---|---|---|
| app-20230124185551-0000 | Big Data 1 | 16 | 1024.0 MiB | | 2023/01/24 18:55:51 | root | FINISHED | 49 s |

**SPARK MASTER**

**SPARK WORKER**

## Spark Worker at 172.18.0.8:41479

**ID:** worker-20230124184613-172.18.0.8-41479
**Master URL:** spark://515c4525e254:7077
**Cores:** 8 (8 Used)
**Memory:** 14.4 GiB (1024.0 MiB Used)
**Resources:**

Back to Master

### Running Executors (1)

| ExecutorID | State | Cores | Memory | Resources | Job Details | Logs |
|---|---|---|---|---|---|---|
| 0 | RUNNING | 8 | 1024.0 MiB | | **ID:** app-20230124225510-0001<br>**Name:** Big Data 1<br>**User:** root | stdout stderr |

### Finished Executors (1)

| ExecutorID | State | Cores | Memory | Resources | Job Details | Logs |
|---|---|---|---|---|---|---|
| 0 | KILLED | 8 | 1024.0 MiB | | **ID:** app-20230124185551-0000<br>**Name:** Big Data 1<br>**User:** root | stdout stderr |

# IZVRŠENJE NA KLASTERU



Spark 3.1.2 **Spark Master at spark://515c4525e254:7077**

**URL:** spark://515c4525e254:7077
**Alive Workers:** 2
**Cores in use:** 16 Total, 0 Used
**Memory in use:** 28.8 GiB Total, 0.0 B Used
**Resources in use:**
**Applications:** 0 Running, 2 Completed
**Drivers:** 0 Running, 0 Completed
**Status:** ALIVE

▸ Workers (2)

▾ Running Applications (0)

| Application ID | Name | Cores | Memory per Executor | Resources Per Executor | Submitted Time | User | State | Duration |
|---|---|---|---|---|---|---|---|---|

▾ Completed Applications (2)

| Application ID | Name | Cores | Memory per Executor | Resources Per Executor | Submitted Time | User | State | Duration |
|---|---|---|---|---|---|---|---|---|
| app-20230124225510-0001 | Big Data 1 | 16 | 1024.0 MiB | | 2023/01/24 22:55:10 | root | FINISHED | 1.1 min |
| app-20230124185551-0000 | Big Data 1 | 16 | 1024.0 MiB | | 2023/01/24 18:55:51 | root | FINISHED | 49 s |

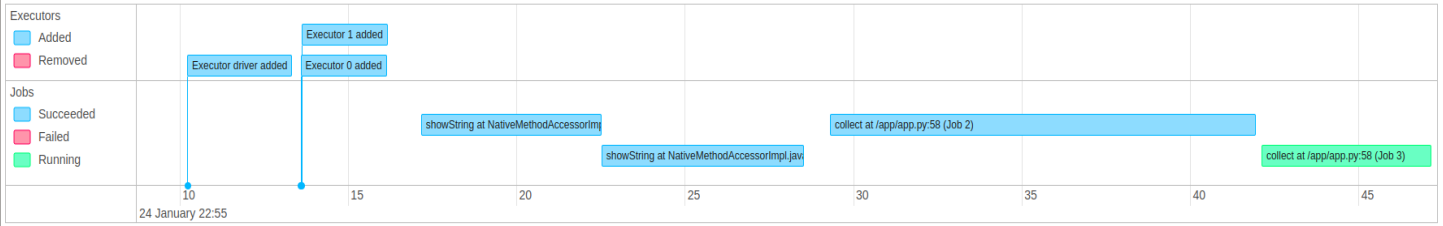
**SPARK MASTER**

**SPARK JOB RUNNING**

**Spark Jobs** (?)

**User:** root
**Total Uptime:** 38 s
**Scheduling Mode:** FIFO
**Active Jobs:** 1
**Completed Jobs:** 3

▾ Event Timeline
☐ Enable zooming



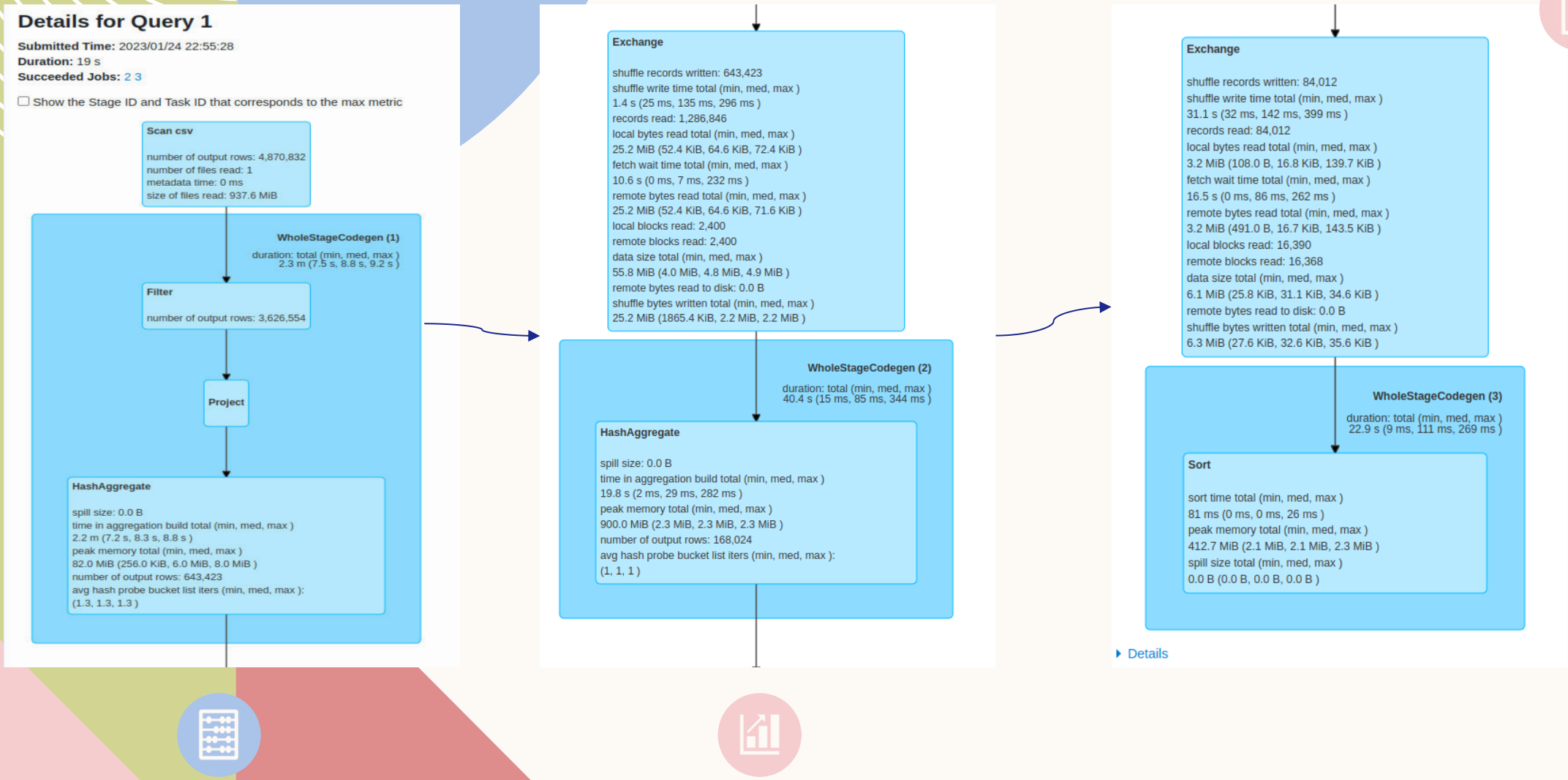

▾ Active Jobs (1)

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

| Job Id ▾ | Description | Submitted | Duration | Stages: Succeeded/Total | Tasks (for all stages): Succeeded/Total |
|---|---|---|---|---|---|
| 3 | collect at /app/app.py:58<br>collect at /app/app.py:58 (kill) | 2023/01/24 22:55:42 | 5 s | 1/3 | 279/416 (17 running) |

# IZVRŠAVANJE TASK-A 2





# DETALJI IZVRŠENJA TASK-A 2



```
== Physical Plan ==
* Sort (8)
+- Exchange (7)
   +- * HashAggregate (6)
      +- Exchange (5)
         +- * HashAggregate (4)
            +- * Project (3)
               +- * Filter (2)
                  +- Scan csv  (1)


(1) Scan csv
Output [6]: [tripduration#0, starttime#1, start_station_name#4, start_station_latitude#5, end_station_name#8, end_station_longitude#10]
Batched: false
Location: InMemoryFileIndex [hdfs://namenode:9000/dir/nyc_bike_rides.csv]
PushedFilters: [IsNotNull(start_station_latitude), IsNotNull(end_station_longitude), GreaterThanOrEqual(start_station_latitude,40.692066), LessThanOrEqual(end_station_longitude,73.77683)]
ReadSchema: struct<tripduration:int,starttime:string,start_station_name:string,start_station_latitude:float,end_station_name:string,end_station_longitude:float>

(2) Filter [codegen id : 1]
Input [6]: [tripduration#0, starttime#1, start_station_name#4, start_station_latitude#5, end_station_name#8, end_station_longitude#10]
Condition : ((((isnotnull(start_station_latitude#5) AND isnotnull(end_station_longitude#10)) AND (start_station_latitude#5 >= 40.692066)) AND (end_station_longitude#10 <= 73.77683)) AND (gettimestamp(starttime#1, yyyy-MM-dd HH:mm:ss,
Some(GMT), false) > 1376525160000000))

(3) Project [codegen id : 1]
Output [3]: [tripduration#0, start_station_name#4, end_station_name#8]
Input [6]: [tripduration#0, starttime#1, start_station_name#4, start_station_latitude#5, end_station_name#8, end_station_longitude#10]

(4) HashAggregate [codegen id : 1]
Input [3]: [tripduration#0, start_station_name#4, end_station_name#8]
Keys [2]: [start_station_name#4, end_station_name#8]
Functions [1]: [partial_stddev_samp(cast(tripduration#0 as double))]
Aggregate Attributes [3]: [n#216, avg#217, m2#218]
Results [5]: [start_station_name#4, end_station_name#8, n#221, avg#222, m2#223]

(5) Exchange
Input [5]: [start_station_name#4, end_station_name#8, n#221, avg#222, m2#223]
Arguments: hashpartitioning(start_station_name#4, end_station_name#8, 200), ENSURE_REQUIREMENTS, [id=#43]

(6) HashAggregate [codegen id : 2]
Input [5]: [start_station_name#4, end_station_name#8, n#221, avg#222, m2#223]
Keys [2]: [start_station_name#4, end_station_name#8]
Functions [1]: [stddev_samp(cast(tripduration#0 as double))]
Aggregate Attributes [1]: [stddev_samp(cast(tripduration#0 as double))#198]
Results [3]: [start_station_name#4, end_station_name#8, stddev_samp(cast(tripduration#0 as double))#198 AS standard_deviation_time_spent_between_stations#199]

(7) Exchange
Input [3]: [start_station_name#4, end_station_name#8, standard_deviation_time_spent_between_stations#199]
Arguments: rangepartitioning(standard_deviation_time_spent_between_stations#199 ASC NULLS FIRST, 200), ENSURE_REQUIREMENTS, [id=#47]

(8) Sort [codegen id : 3]
Input [3]: [start_station_name#4, end_station_name#8, standard_deviation_time_spent_between_stations#199]
Arguments: [standard_deviation_time_spent_between_stations#199 ASC NULLS FIRST], true, 0
```

Izvršenje task-a kojim se određuje ruta na kojoj je standardna devijacija vremena vožnje bila najmanja

# OPIS PROJEKTA 2



- Dataset koji se nalazi na hdfs-u se sada koristi tako da se podaci skladišteni na namenode-u šalju na određeni Kafka topic (jedna poslata poruka odgovara jednom redu dataset-a)
- Osluškuje se odgovarajući Kafka topic i pristigle poruke se dalje obrađuju korišćenjem Pyspark/Flink biblioteke vezane za tokove podataka (streaming)
- Obrada se vrši nad svim porukama pristiglim u definisanom vremenskom prozoru (2 sekunde)
- Obrada podrazumeva primenu osnovnih funkcija agregacije i filtriranja
- Traženje broja pojava datog parametra u podacima pristiglim u datom vremenskom prozoru
- Obrađeni podaci upisuju se u NoSQL bazu podataka Cassandra
- Izvršenje na Spark klasteru i izvršenje na Flink klasteru

# IMPLEMENTACIJA PROJEKTA 2

Značajni delovi

# STREAM OBRADA PODATAKA



## PYSPARK

- Kafka Producer i Consumer napisani su u Python-u
- Streaming obrada podataka vrši se nad vrstama dataset-a (poslate na Kafka topic) u vremenskim prozorima od 2 sekunde
- Posmatraju se samo vožnje koje su završene na nekoj Aveniji
- Izračunavanje max i min vremena putovanja, standardne devijacije istog, kao i ukupno vreme trajanja svih vožnji
- Pronalaženje tri stanice na kojima su putnici najčešće završavali vožnju
- Upis svih rezultata u Cassandra-u
- Obrada na Spark-u kroz objekat Streaming Context-a (Kafka Consumer aplikacija)

## FLINK

- Kafka Producer napisan u Python-u, Kafka Consumer u Java-i
- Streaming obrada podataka vrši se nad vrstama dataset-a (poslate na Kafka topic) u vremenskim prozorima od 2 sekunde
- Posmatraju se samo vožnje koje su završene na nekoj Aveniji
- Izračunavanje max i min vremena putovanja, standardne devijacije istog, kao i ukupno vreme trajanja svih vožnji
- Pronalaženje tri stanice na kojima su putnici najčešće završavali vožnju
- Upis svih rezultata u Cassandra-u
- Obrada na Flink-u nakon prosleđivanja posla Job Manager-u kroz .jar fajl (Kafka Consumer aplikacija)

# IZVRŠENJE NA SPARK KLASTERU



Spark 2.4.3 **Spark Master at spark://698ca3ac0e57:7077**

**URL:** spark://698ca3ac0e57:7077
**Alive Workers:** 2
**Cores in use:** 16 Total, 16 Used
**Memory in use:** 28.8 GB Total, 2.0 GB Used
**Applications:** 1 Running, 0 Completed
**Drivers:** 0 Running, 0 Completed
**Status:** ALIVE

**Workers (2)**

| Worker Id | Address | State | Cores | Memory |
|---|---|---|---|---|
| worker-20230227225303-172.20.0.10-39831 | 172.20.0.10:39831 | ALIVE | 8 (8 Used) | 14.4 GB (1024.0 MB Used) |
| worker-20230227225303-172.20.0.9-37805 | 172.20.0.9:37805 | ALIVE | 8 (8 Used) | 14.4 GB (1024.0 MB Used) |

**Running Applications (1)**

| Application ID | Name | Cores | Memory per Executor | Submitted Time | User | State | Duration |
|---|---|---|---|---|---|---|---|
| app-20230227225423-0000 (kill) | Big Data 2 | 16 | 1024.0 MB | 2023/02/27 22:54:23 | root | RUNNING | 31 s |

**SPARK MASTER**

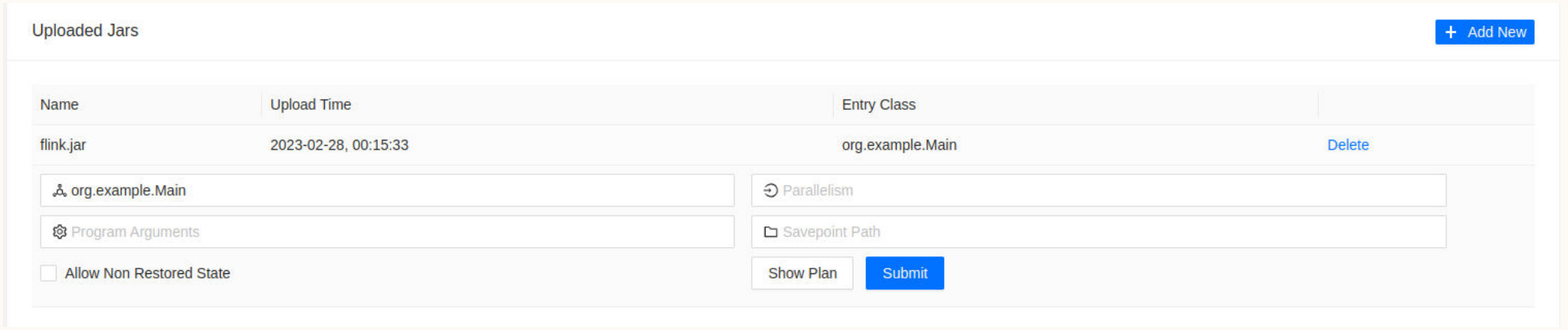
**RESULTS**

```
Trip duration parameters: max=1240.0     min=1201.0     standard deviation=19.5         count=2
Trip duration parameters: max=646.0      min=646.0      standard deviation=0.0  count=1
Trip duration parameters: max=660.0      min=388.0      standard deviation=136.0        count=2
Trip duration parameters: max=1147.0     min=431.0      standard deviation=308.4632158873332    count=3
Trip duration parameters: max=1356.0     min=279.0      standard deviation=491.19061699326284   count=3
Trip duration parameters: max=2158.0     min=690.0      standard deviation=570.8639833620614    count=4
Trip duration parameters: max=377.0      min=377.0      standard deviation=0.0  count=1
Trip duration parameters: max=309.0      min=309.0      standard deviation=0.0  count=1
Trip duration parameters: max=448.0      min=448.0      standard deviation=0.0  count=1
Trip duration parameters: max=589.0      min=589.0      standard deviation=0.0  count=1
```

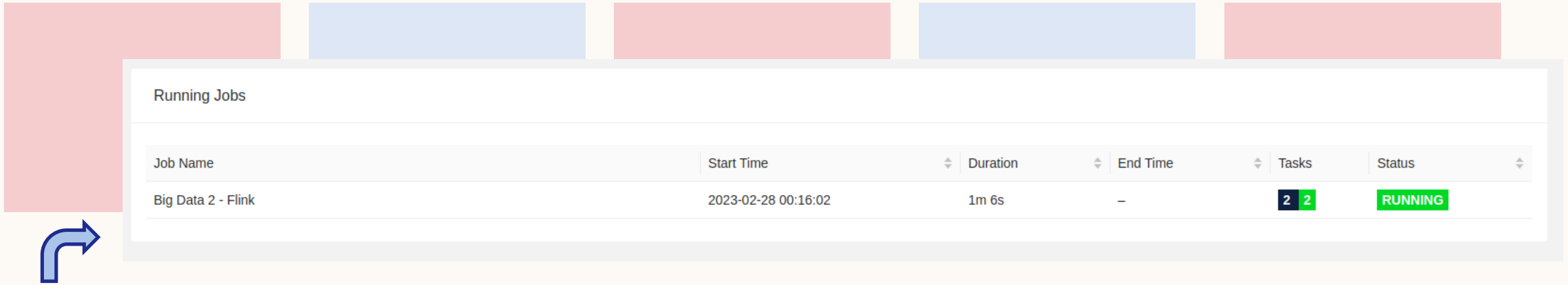
```
23/02/27 22:56:02 INFO DAGScheduler: ResultStage 198 (runJob at PythonRDD.scala:153) finished in 0.056 s
23/02/27 22:56:02 INFO DAGScheduler: Job 149 finished: runJob at PythonRDD.scala:153, took 0.127721 s
[('Lafayette Ave & Fort Greene Pl', 1), ('Lafayette St & E 8 St', 1), ('E 51 St & Lexington Ave', 1)]
-------------------------------------------------------------------------------------------
23/02/27 22:56:02 INFO JobScheduler: Finished job streaming job 1677538562000 ms.0 from job set of time 1677538562000 ms
23/02/27 22:56:02 INFO JobScheduler: Total delay: 0.296 s for time 1677538562000 ms (execution: 0.284 s)
```

# IZVRŠENJE NA FLINK KLASTERU






**SUBMIT JOB**




**JOB MANAGER**

# IZVRŠENJE NA FLINK KLASTERU

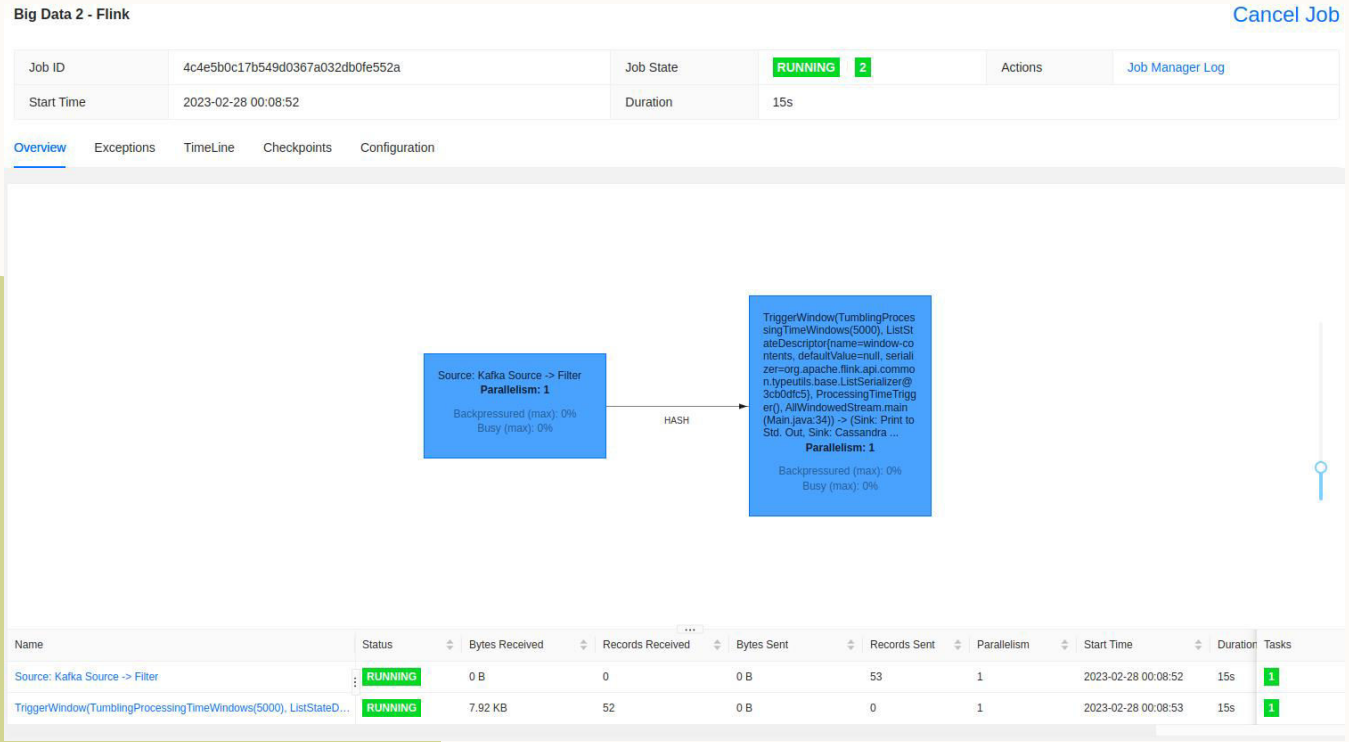


TASK MANAGER

Task Managers

| Path, ID | Data Port | Last Heartbeat | All Slots | Free Slots | CPU Cores | Physical MEM | JVM Heap Size | Flink Managed MEM |
|---|---|---|---|---|---|---|---|---|
| 192.168.16.7:45833-d0add0<br>akka.tcp://flink@192.168.16.7:45833/user/rpc/taskmanager_0 | 43341 | 2023-02-28 00:10:24 | 1 | 1 | 8 | 15.4 GB | 512 MB | 512 MB |
| 192.168.16.8:40655-1409b1<br>akka.tcp://flink@192.168.16.8:40655/user/rpc/taskmanager_0 | 46247 | 2023-02-28 00:10:24 | 1 | 0 | 8 | 15.4 GB | 512 MB | 512 MB |

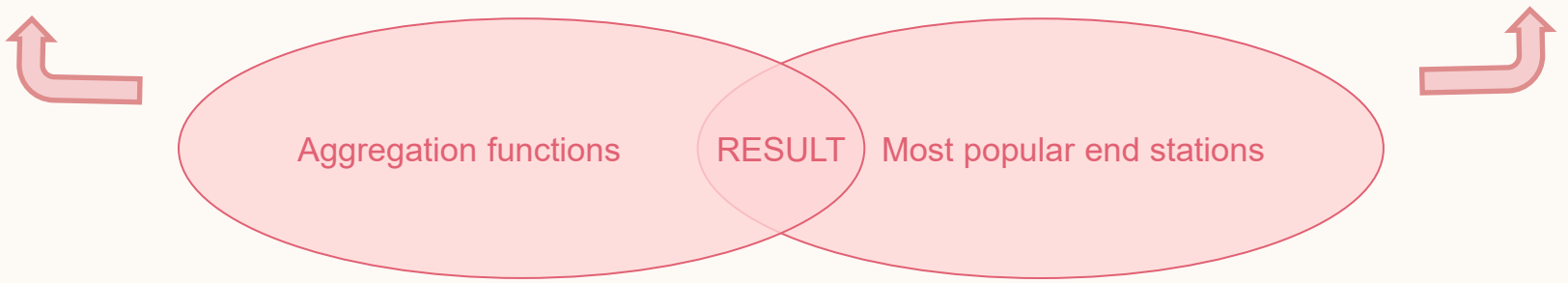
RUNNING JOB



# IZVRŠENJE NA FLINK KLASTERU



```
min: 240.0max: 2339.0avg: 699.9167stddev: 424.90251986239616W 37 St & 5 Ave: 1E 48 St & 3 Ave: 2W 56 St & 6 Ave: 1date: Mon Feb 27 23:11:10 UTC 2023
min: 120.0max: 1943.0avg: 733.08stddev: 459.2714967121169Adelphi St & Myrtle Ave: 1E 23 St & 1 Ave: 1E 55 St & 2 Ave: 1date: Mon Feb 27 23:11:15 UTC 2023
min: 185.0max: 3559.0avg: 756.2286stddev: 619.1602762172234W 37 St & 5 Ave: 1W 54 St & 9 Ave: 1E 32 St & Park Ave: 1date: Mon Feb 27 23:11:20 UTC 2023
min: 90.0max: 2241.0avg: 665.9677stddev: 505.7745456946547E 56 St & 3 Ave: 1DeKalb Ave & Vanderbilt Ave: 1W 43 St & 6 Ave: 1date: Mon Feb 27 23:11:25 UTC 2023
min: 137.0max: 2204.0avg: 816.2222stddev: 579.4159616155896E 47 St & 2 Ave: 1E 32 St & Park Ave: 1W 56 St & 6 Ave: 2date: Mon Feb 27 23:11:30 UTC 2023
min: 109.0max: 1686.0avg: 603.74194stddev: 334.6943687896164W 37 St & 5 Ave: 2Central Park S & 6 Ave: 1W 43 St & 6 Ave: 1date: Mon Feb 27 23:11:35 UTC 2023
min: 198.0max: 2158.0avg: 742.30304stddev: 439.3054829487503E 47 St & 2 Ave: 1E 32 St & Park Ave: 1W 56 St & 6 Ave: 1date: Mon Feb 27 23:11:40 UTC 2023
min: 295.0max: 1921.0avg: 765.3333stddev: 485.06617116646737W 51 St & 6 Ave: 1E 32 St & Park Ave: 1E 48 St & 3 Ave: 2date: Mon Feb 27 23:11:45 UTC 2023
min: 143.0max: 1972.0avg: 711.5769stddev: 500.58597763612744E 32 St & Park Ave: 2E 48 St & 3 Ave: 1W 51 St & 6 Ave: 1date: Mon Feb 27 23:11:50 UTC 2023
min: 190.0max: 1572.0avg: 776.88464stddev: 422.28229046484705E 48 St & 3 Ave: 1E 47 St & 2 Ave: 2W 51 St & 6 Ave: 1date: Mon Feb 27 23:11:55 UTC 2023
min: 198.0max: 1751.0avg: 832.4643stddev: 408.6996349174204W 37 St & 5 Ave: 1E 32 St & Park Ave: 1W 51 St & 6 Ave: 1date: Mon Feb 27 23:12:00 UTC 2023
min: 287.0max: 1889.0avg: 875.7308stddev: 385.3344758570451W 56 St & 6 Ave: 1W 21 St & 6 Ave: 2W 25 St & 6 Ave: 1date: Mon Feb 27 23:12:05 UTC 2023
min: 251.0max: 2889.0avg: 872.68964stddev: 571.777678393666Carmine St & 6 Ave: 11 Ave & E 15 St: 1W 21 St & 6 Ave: 1date: Mon Feb 27 23:12:10 UTC 2023
```



# ZNAČAJNI DELOVI KODA – SPARK I FLINK



```python
msgProducer = KafkaProducer(bootstrap_servers=['localhost:50023'], value_serializer = lambda x: x.encode('utf-8'))

with open('data//nyc_bike_rides.csv') as csvFile:
    data = csv.DictReader(csvFile)
    for row in data:
        msgProducer.send('test', json.dumps(row))
        msgProducer.flush()

        print('Message sent with body: ' + json.dumps(row))
        time.sleep(0.1)
```

KAFKA PRODUCER

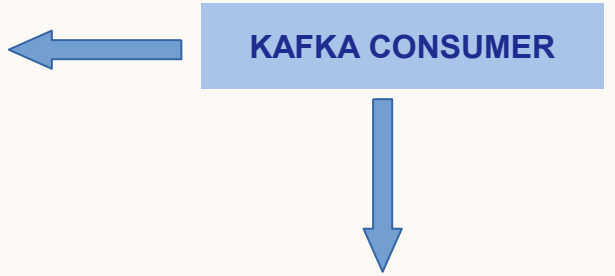
CASSANDRA MANIPULATION

```python
keyspace = 'project2_streaming_keyspace'
cassandra_session.execute("""
    CREATE KEYSPACE IF NOT EXISTS %s
    WITH replication = { 'class': 'SimpleStrategy', 'replication_factor': '1' }
    """ % keyspace)

cassandra_session.set_keyspace(keyspace)

cassandra_session.execute("""
    CREATE TABLE IF NOT EXISTS tripduration (
        time TIMESTAMP,
        end_station text,
        max float,
        min float,
        cnt float,
        std float,
        PRIMARY KEY (time)
    )
    """)

cassandra_session.execute("""
    CREATE TABLE IF NOT EXISTS popular_streets (
        time TIMESTAMP,
        street1 text,
        name1 int,
        street2 text,
        name2 int,
        street3 text,
        name3 int,
        PRIMARY KEY (time)
    )
    """)
```

# ZNAČAJNI DELOVI KODA – SPARK



```
filtered = data.filter(lambda x: (end_station in x['end station name']))

if (filtered.isEmpty()):
    print('Not a single ride ended on any of the Avenues!')
else:
    tripduration_max = filtered.map(lambda x: float(x['tripduration'])).max()
    tripduration_min = filtered.map(lambda x: float(x['tripduration'])).min()
    tripduration_cnt = filtered.map(lambda x: float(x['tripduration'])).count()
    tripduration_std = filtered.map(lambda x: float(x['tripduration'])).stdev()

    job1 = 'Trip duration parameters: max=' + str(tripduration_max) + '\t min=' + str(tripduration_min) + '\t standard deviation=' + str(tripduration_std)\
        + '\t count=' + str(tripduration_cnt)

    print(job1)

    cassandra_session.execute(f"""
    INSERT INTO project2_streaming_keyspace.tripduration(time, end_station, max, min, cnt, std)
    VALUES (toTimeStamp(now()), '{end_station}', {tripduration_max}, {tripduration_min}, {tripduration_cnt}, {tripduration_std})
    """)

popular_streets = data.map(lambda x: (x['end station name'], 1)).reduceByKey(lambda x,y : x+y).sortBy(lambda x: x[1],ascending=False).take(3)

N = len(popular_streets)
if N==0:
    popular_streets = [('',0),('',0),('',0)]
elif N==1:
    popular_streets.append(('',0))
    popular_streets.append(('',0))
elif N==2:
    popular_streets.append(('',0))
```




```
sc = SparkContext(appName='Big Data 2')
streaming = StreamingContext(sc, int(window_duration))

stream = KafkaUtils.createDirectStream(streaming, [topic], {'metadata.broker.list':kafka_url})
print('Connected to Kafka!')

stream.foreachRDD(lambda input: structured_streaming(input, end_station))

streaming.start()
streaming.awaitTermination()
```

# ZNAČAJNI DELOVI KODA – FLINK



KAFKA CONSUMER

```
HashMap<String, Integer> popular = new HashMap<>();

for (MessageFromTopic msg : elements) {
    count ++;
    sum += msg.tripduration;
    if (msg.tripduration > max)
        max = msg.tripduration;
    if (msg.tripduration < min)
        min = msg.tripduration;
    if(!popular.containsKey(msg.end_station_name)) {
        popular.put(msg.end_station_name, v: 1);
    } else {
        int newValue = popular.get(msg.end_station_name) + 1;
        popular.replace(msg.end_station_name, newValue);
    }
}
avg = sum / count;

street1 = (String) popular.keySet().toArray()[0];
name1 = popular.get(street1);
street2 = (String) popular.keySet().toArray()[1];
name2 = popular.get(street2);
street3 = (String) popular.keySet().toArray()[2];
name3 = popular.get(street3);

for (MessageFromTopic msg : elements) {
    stddev += Math.pow(msg.tripduration - avg, 2);
}

stddev = Math.sqrt(stddev / count);
Date date = new Date();

TripDurationStatistics res = new TripDurationStatistics(min, max, avg, stddev, street1, name1, street2, name2, street3, name3, date);
  System.out.println("final res ---> " + res);
out.collect(res);
```

```
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
```

```
DataStream<MessageFromTopic> ds = env.fromSource(source, WatermarkStrategy.noWatermarks(), sourceName: "Kafka Source").filter((FilterFunction<MessageFromTopic>) value -> (value.end_station_name.contains("Ave")));
DataStream<TripDurationStatistics> res = ds.windowAll(TumblingProcessingTimeWindows.of(Time.seconds(5))).process(new StatisticsStream());
```

```
env.setParallelism(2);
env.execute( jobName: "Big Data 2 - Flink");
```

# OPIS PROJEKTA 3

- Pored funkcija iz biblioteke Pyspark koje su pomenute u prvom i drugom projektu koriste se i funkcije vezane za ML (mašinsko učenje)
- Treniranje podataka iz postojećeg dataset-a
- Model podataka dobijen na osnovu feature-a (latituda i longituda početne stanice)
- Kreiranje Pipeline-ova za obradu podataka
- Klasifikacija dobijenog modela podataka
- Kafka Producer aplikacija korišćena je iz projekta 2
- Rezultati predikcije kao i podaci sa Kafka topic-a (sample-ovani podaci za vremenski okvir – 2s) upisuju se u bazu podataka InfluxDB
- Vizuelizacija podataka kroz Grafana-u

# IMPLEMENTACIJA PROJEKTA 3

Značajni delovi

# IZVRŠENJE NA SPARK KLASTERU



## Spark Master at spark://0f78be9868e9:7077

**URL:** spark://0f78be9868e9:7077
**Alive Workers:** 2
**Cores in use:** 16 Total, 16 Used
**Memory in use:** 28.8 GB Total, 2.0 GB Used
**Applications:** 1 Running, 1 Completed
**Drivers:** 0 Running, 0 Completed
**Status:** ALIVE

### Workers (2)

| Worker Id | Address | State | Cores | Memory |
|---|---|---|---|---|
| worker-20230228223524-172.20.0.11-45413 | 172.20.0.11:45413 | ALIVE | 8 (8 Used) | 14.4 GB (1024.0 MB Used) |
| worker-20230228223524-172.20.0.9-38051 | 172.20.0.9:38051 | ALIVE | 8 (8 Used) | 14.4 GB (1024.0 MB Used) |

### Running Applications (1)

| Application ID | Name | Cores | Memory per Executor | Submitted Time | User | State | Duration |
|---|---|---|---|---|---|---|---|
| app-20230228224255-0001 (kill) | Big Data 3 - Classification | 16 | 1024.0 MB | 2023/02/28 22:42:55 | root | RUNNING | 6 s |

### Completed Applications (1)

| Application ID | Name | Cores | Memory per Executor | Submitted Time | User | State | Duration |
|---|---|---|---|---|---|---|---|
| app-20230228223606-0000 | Big Data 3 - Training | 16 | 1024.0 MB | 2023/02/28 22:36:06 | root | FINISHED | 4.5 min |

**SPARK MASTER**

**MODEL ON HDFS**

| Permission | Owner | Group | Size | Last Modified | Replication | Block Size | Name |
|---|---|---|---|---|---|---|---|
| -rw-r--r-- | root | supergroup | 0 B | Feb 28 23:40 | 3 | 128 MB | _SUCCESS |
| -rw-r--r-- | root | supergroup | 212 B | Feb 28 23:40 | 3 | 128 MB | part-00000 |

Showing 1 to 2 of 2 entries

Previous 1 Next

Hadoop, 2019.

# ZNAČAJNI DELOVI KODA



**CLASSIFICATION**

```
dataFrame = data.toDF()

columns = ['start station latitude', 'start station longitude']

for column in columns:
    dataFrame = dataFrame.withColumn(column, F.col(column).cast(FloatType()))

vectorAssembler = VectorAssembler().setInputCols(columns).setOutputCol('features').setHandleInvalid(

assembled = vectorAssembler.transform(dataFrame)

stringIndexer = StringIndexer().setInputCol('usertype').setOutputCol('label')
indexedDataFrame = stringIndexer.fit(assembled).transform(assembled)

prediction = model.transform(indexedDataFrame)

prediction.select('prediction', 'label')

predictionsMade = prediction.count()

correctNumber = float(prediction.filter(prediction['label'] == prediction['prediction']).count())
```

**TRAINING**

```
spark = SparkSession.builder.appName('Big Data 3 - Training').getOrCreate()
spark.sparkContext.setLogLevel("ERROR")

HDFS_DATA = os.getenv('HDFS_URL')
DATASET = os.getenv('DATASET_LOCATION')
MODEL = os.getenv('MODEL_LOCATION')

dataFrame = spark.read.csv(HDFS_DATA + DATASET, header=True)

columns = ['start station latitude', 'start station longitude']

for column in columns:
    dataFrame = dataFrame.withColumn(column, F.col(column).cast(FloatType()))

vectorAssembler = VectorAssembler().setInputCols(columns).setOutputCol('features').setHandleInvalid('skip')

assembled = vectorAssembler.transform(dataFrame)

stringIndexer = StringIndexer().setInputCol('usertype').setOutputCol('label')
indexedDataFrame = stringIndexer.fit(assembled).transform(assembled)

train_split, test_split = indexedDataFrame.randomSplit([0.8, 0.2], seed=1337)

regressionModel = LogisticRegression(maxIter=100, regParam=0.02, elasticNetParam=0.8)

pipeline = Pipeline(stages=[regressionModel])
regressionModelPipe = pipeline.fit(train_split)

prediction = regressionModelPipe.transform(test_split)

evaluator = BinaryClassificationEvaluator(labelCol='label', rawPredictionCol='prediction', metricName='areaUnderROC')
accuracy = evaluator.evaluate(prediction)

print('Accuracy\'s value for logistic regression model is ' + str(accuracy) + '!')

regressionModelPipe.write().overwrite().save(HDFS_DATA + MODEL)
```

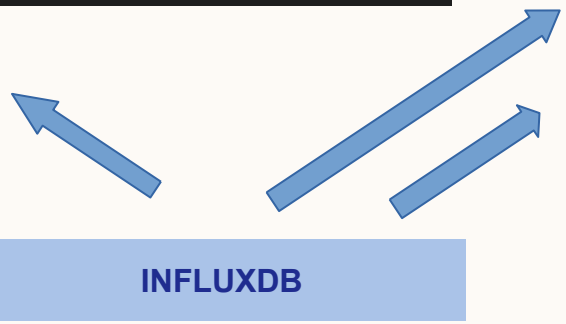
# ZNAČAJNI DELOVI KODA



```python
def influxDBconnect():
    return InfluxDBClient(dbhost, dbport, dbuser, dbpassword, dbname)

def influxDBwrite(count, accuracy, topic, data):
    timestamp = datetime.utcnow().strftime('%Y-%m-%dT%H:%M:%SZ')
    measurementData = [
        {
            "measurement": topic,
            "time": timestamp,
            "fields": {
                "start station latitude": data[0],
                "start station longitude": data[1],
                "end station latitude": data[2],
                "end station longitude": data[3],
                "trip duration": data[4],
                "predictions": count,
                "accuracy": accuracy
            }
        }
    ]
    print(measurementData)
    influxDBConnection.write_points(measurementData, time_precision='ms')
```

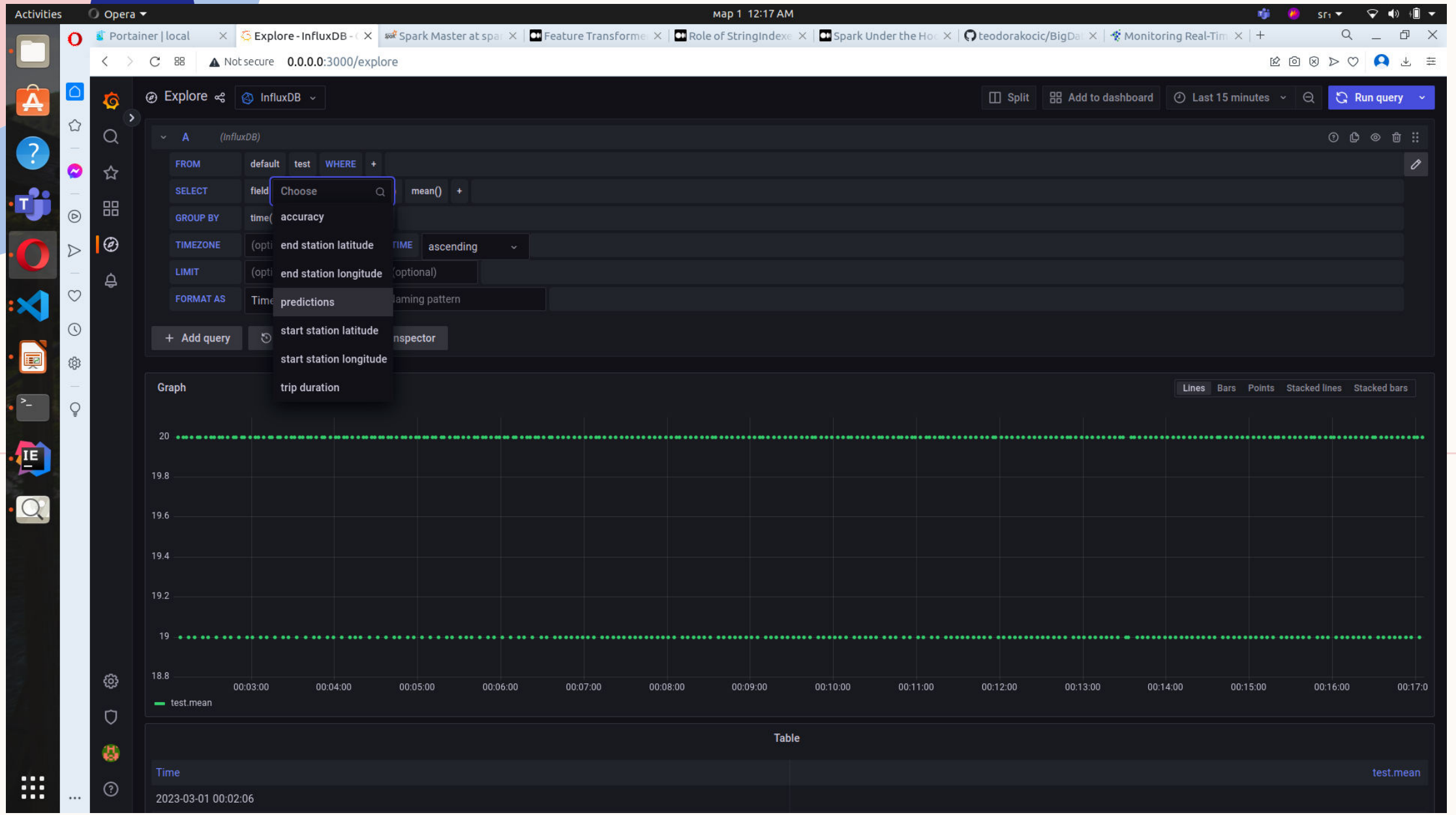
```python
forBucket = list()

if(data.isEmpty()):
    print('No data available')
    return
else:
    sample = data.toDF().count() // 2
    forBucket.append(float(data.toDF().collect()[sample][3]))
    forBucket.append(float(data.toDF().collect()[sample][4]))
    forBucket.append(float(data.toDF().collect()[sample][8]))
    forBucket.append(float(data.toDF().collect()[sample][9]))
    forBucket.append(int(data.toDF().collect()[sample][13]))
```

```python
influxDBwrite(predictionsMade, correctNumber, topic, forBucket)
```



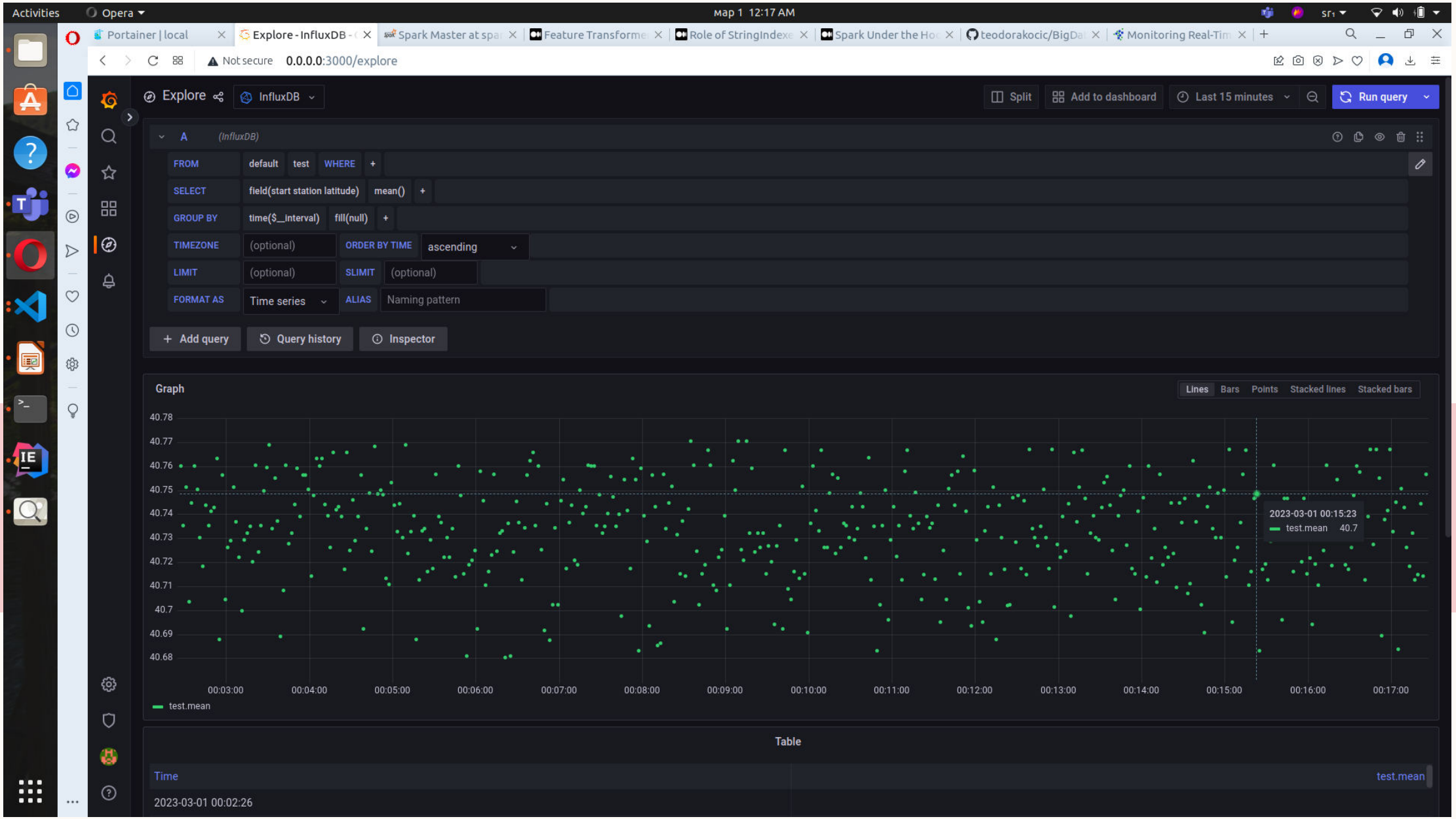
# VIZUELIZACIJA – GRAFANA





# VIZUELIZACIJA – GRAFANA





# LITERATURA


1. Slajdovi sa predavanja i vežbi
2. https://github.com/jonesberg/DataAnalysisWithPythonAndPySpark, Jonathan Rioux
3. https://selectfrom.dev/apache-spark-structured-streaming-via-docker-compose-3e1f146384b9
4. https://nightlies.apache.org/flink/flink-docs-release-1.16/docs/dev/datastream/overview/
5. https://www.javacodegeeks.com/2016/11/monitoring-real-time-uber-data-using-spark-machine-learning-streaming-kafka-api-part-1.html