Miloš Veljanovski 1559

Projekat 1 – Sistemi za analizu velike količine podataka

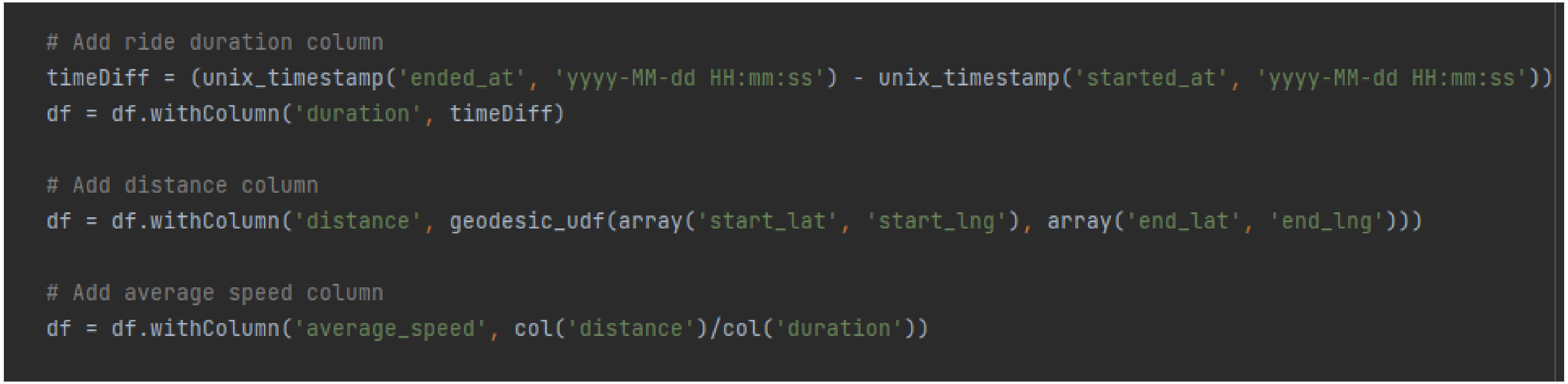
Korišćeni izvorni podaci

- Iskorišćen je set podataka o vožnji biciklama iz „oblasti zaliva“ tj. San Francisko zaliva (uključujući Palo Alto i San Hoze)
- Dataset sadrži sledeće kolene podataka:
  - Vreme i datum početka i završetka vožnje
  - Podaci o početkoj stanici (ID, ime, geografska širina i dužina)
  - Podaci o završnoj stanici (ID, ime, geografska širina i dužina)
  - ID vožnje
  - Tip korisnika (vozača)
- Korišćeni su podaci za 2021. i 2022. godinu

Samostalna python aplikacija koja koristi Spark biblioteku

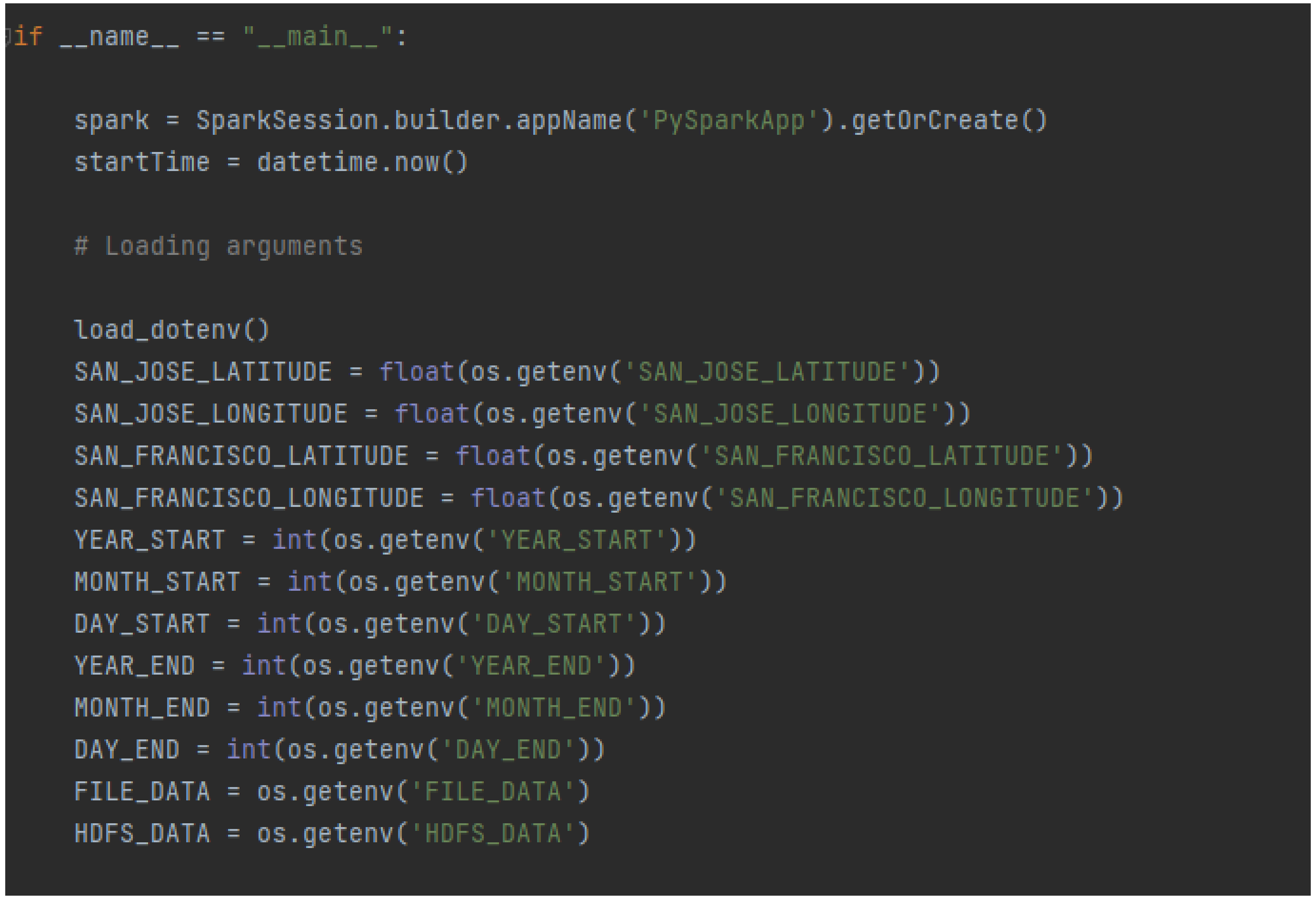
main.py

- Inicijalnom setu podataka sam izračunao i dodao sledeće kolone: vreme trajanja vožnje, pređena distanca (tačnije udaljenost početne i kranje stanice vazdušnom linijom) i srednja brzina



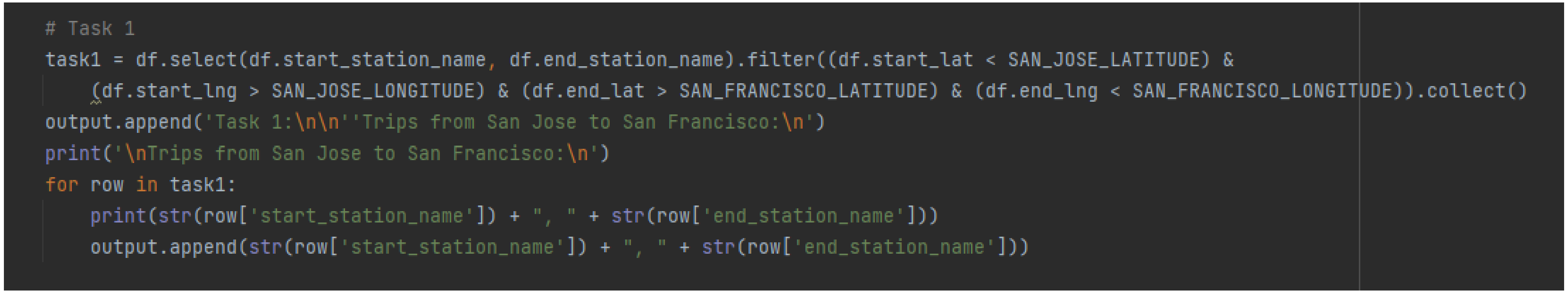
```
# Add ride duration column
timeDiff = (unix_timestamp('ended_at', 'yyyy-MM-dd HH:mm:ss') - unix_timestamp('started_at', 'yyyy-MM-dd HH:mm:ss'))
df = df.withColumn('duration', timeDiff)

# Add distance column
df = df.withColumn('distance', geodesic_udf(array('start_lat', 'start_lng'), array('end_lat', 'end_lng')))

# Add average speed column
df = df.withColumn('average_speed', col('distance')/col('duration'))
```

- Ulazne argumente aplikacije učitavam koristeći dotenv biblioteku



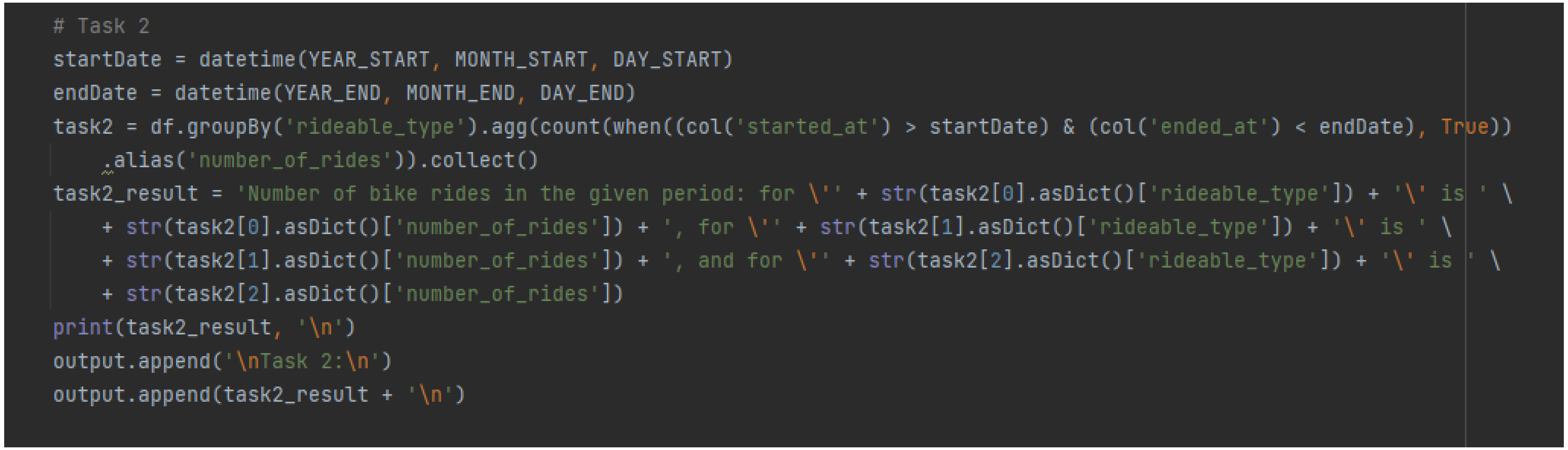
```
if __name__ == "__main__":

    spark = SparkSession.builder.appName('PySparkApp').getOrCreate()
    startTime = datetime.now()

    # Loading arguments

    load_dotenv()
    SAN_JOSE_LATITUDE = float(os.getenv('SAN_JOSE_LATITUDE'))
    SAN_JOSE_LONGITUDE = float(os.getenv('SAN_JOSE_LONGITUDE'))
    SAN_FRANCISCO_LATITUDE = float(os.getenv('SAN_FRANCISCO_LATITUDE'))
    SAN_FRANCISCO_LONGITUDE = float(os.getenv('SAN_FRANCISCO_LONGITUDE'))
    YEAR_START = int(os.getenv('YEAR_START'))
    MONTH_START = int(os.getenv('MONTH_START'))
    DAY_START = int(os.getenv('DAY_START'))
    YEAR_END = int(os.getenv('YEAR_END'))
    MONTH_END = int(os.getenv('MONTH_END'))
    DAY_END = int(os.getenv('DAY_END'))
    FILE_DATA = os.getenv('FILE_DATA')
    HDFS_DATA = os.getenv('HDFS_DATA')
```

- U svrhu demonstracije, kroz aplikaciju sam realizovao 5 zadataka (task-ova) za smislenu obradu podataka
- Zadatak 1 prikazuje imena početne i krajnje stanice svih vožnji koje su počele u San Hoze-u (početne koordinate su južnije od 37.43749 i istočnije od -122.08746) i završile se u San Francisku (krajnje koordinate su severnije od 37.65876 i zapadnije od -122.36854)



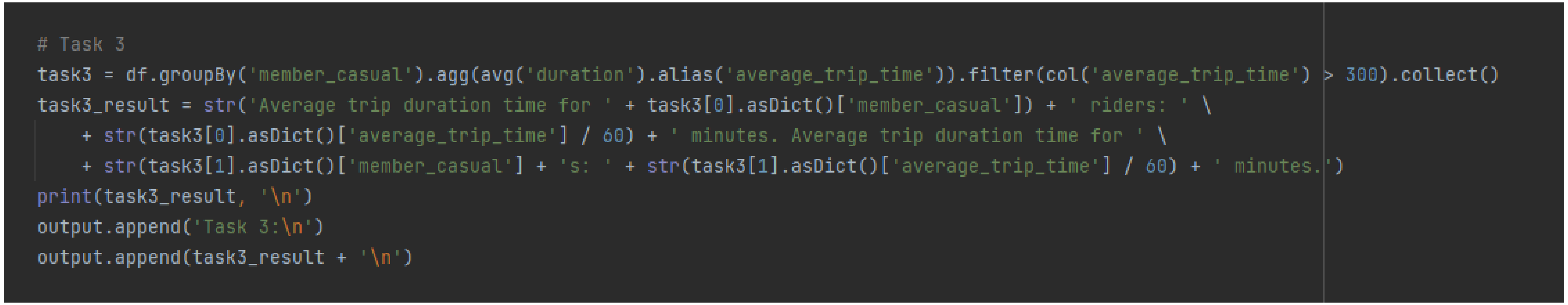
```
# Task 1
task1 = df.select(df.start_station_name, df.end_station_name).filter((df.start_lat < SAN_JOSE_LATITUDE) &
    (df.start_lng > SAN_JOSE_LONGITUDE) & (df.end_lat > SAN_FRANCISCO_LATITUDE) & (df.end_lng < SAN_FRANCISCO_LONGITUDE)).collect()
output.append('Task 1:\n\n''Trips from San Jose to San Francisco:\n')
print('\nTrips from San Jose to San Francisco:\n')
for row in task1:
    print(str(row['start_station_name']) + ", " + str(row['end_station_name']))
    output.append(str(row['start_station_name']) + ", " + str(row['end_station_name']))
```

- Zadatak 2 prikazuje ukupan broj vožnji za svaki tip bicikle za dati vremenski period (od 01.06.2022. do 05.06.2022.)



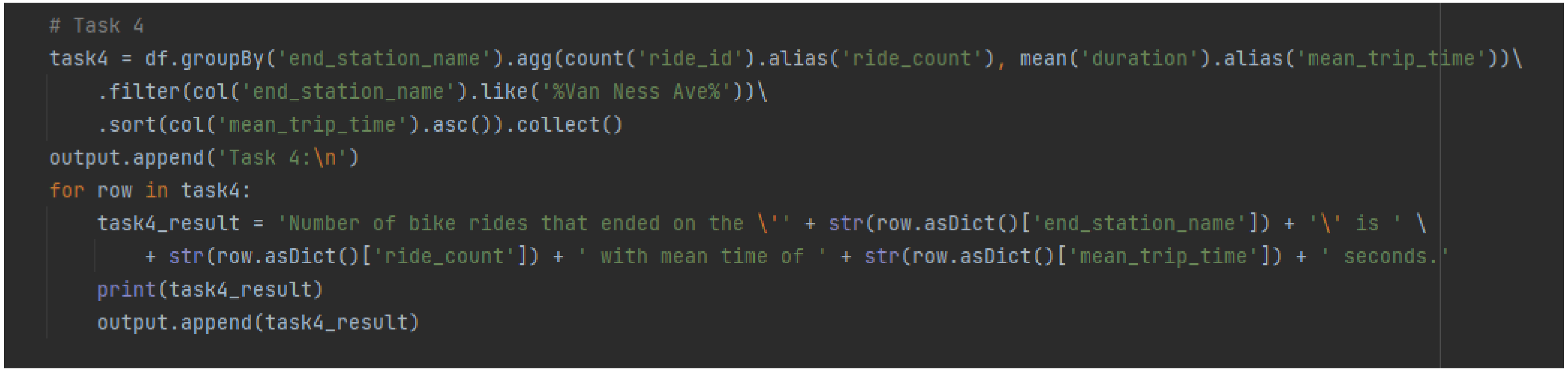
```
# Task 2
startDate = datetime(YEAR_START, MONTH_START, DAY_START)
endDate = datetime(YEAR_END, MONTH_END, DAY_END)
task2 = df.groupBy('rideable_type').agg(count(when((col('started_at') > startDate) & (col('ended_at') < endDate), True))
    .alias('number_of_rides')).collect()
task2_result = 'Number of bike rides in the given period: for \'' + str(task2[0].asDict()['rideable_type']) + '\' is ' \
    + str(task2[0].asDict()['number_of_rides']) + ', for \'' + str(task2[1].asDict()['rideable_type']) + '\' is ' \
    + str(task2[1].asDict()['number_of_rides']) + ', and for \'' + str(task2[2].asDict()['rideable_type']) + '\' is ' \
    + str(task2[2].asDict()['number_of_rides'])
print(task2_result, '\n')
output.append('\nTask 2:\n')
output.append(task2_result + '\n')
```

- Zadatak 3 prikazuje prosečno trajanje vožnje za oba tipa korisnika



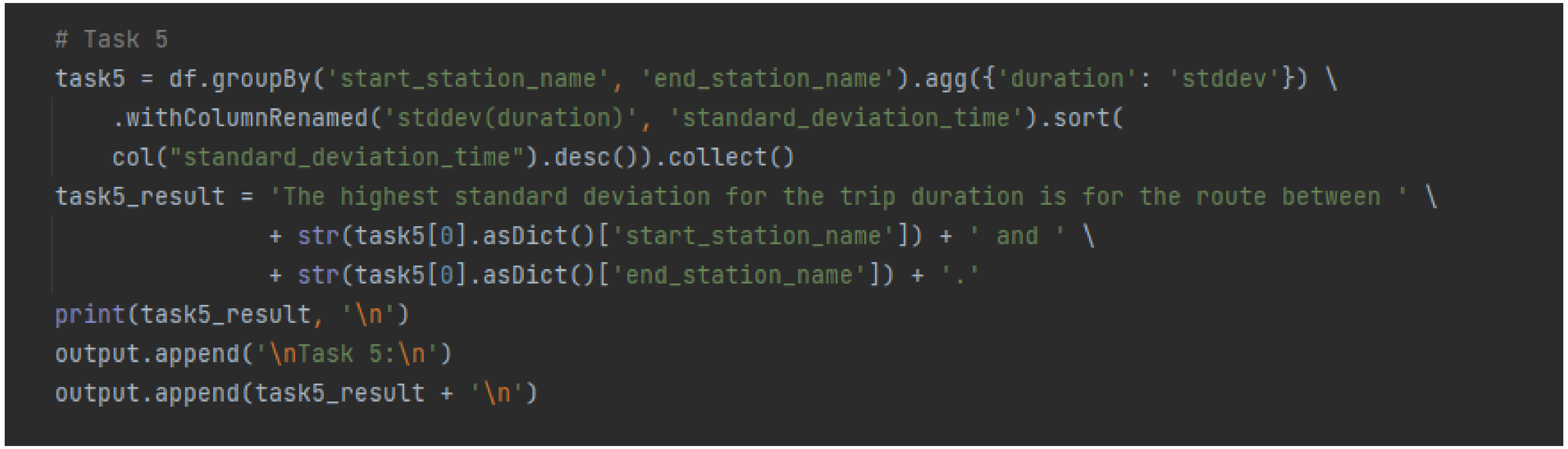
```
# Task 3
task3 = df.groupBy('member_casual').agg(avg('duration').alias('average_trip_time')).filter(col('average_trip_time') > 300).collect()
task3_result = str('Average trip duration time for ' + task3[0].asDict()['member_casual']) + ' riders: ' \
    + str(task3[0].asDict()['average_trip_time'] / 60) + ' minutes. Average trip duration time for ' \
    + str(task3[1].asDict()['member_casual'] + 's: ' + str(task3[1].asDict()['average_trip_time'] / 60) + ' minutes.')
print(task3_result, '\n')
output.append('Task 3:\n')
output.append(task3_result + '\n')
```

- Zadatak 4 prikazuje broj biciklista koji su završili vožnju na nekoj od stanica na Van Nes aveniji i srednja vremena trajanja tih vožnji



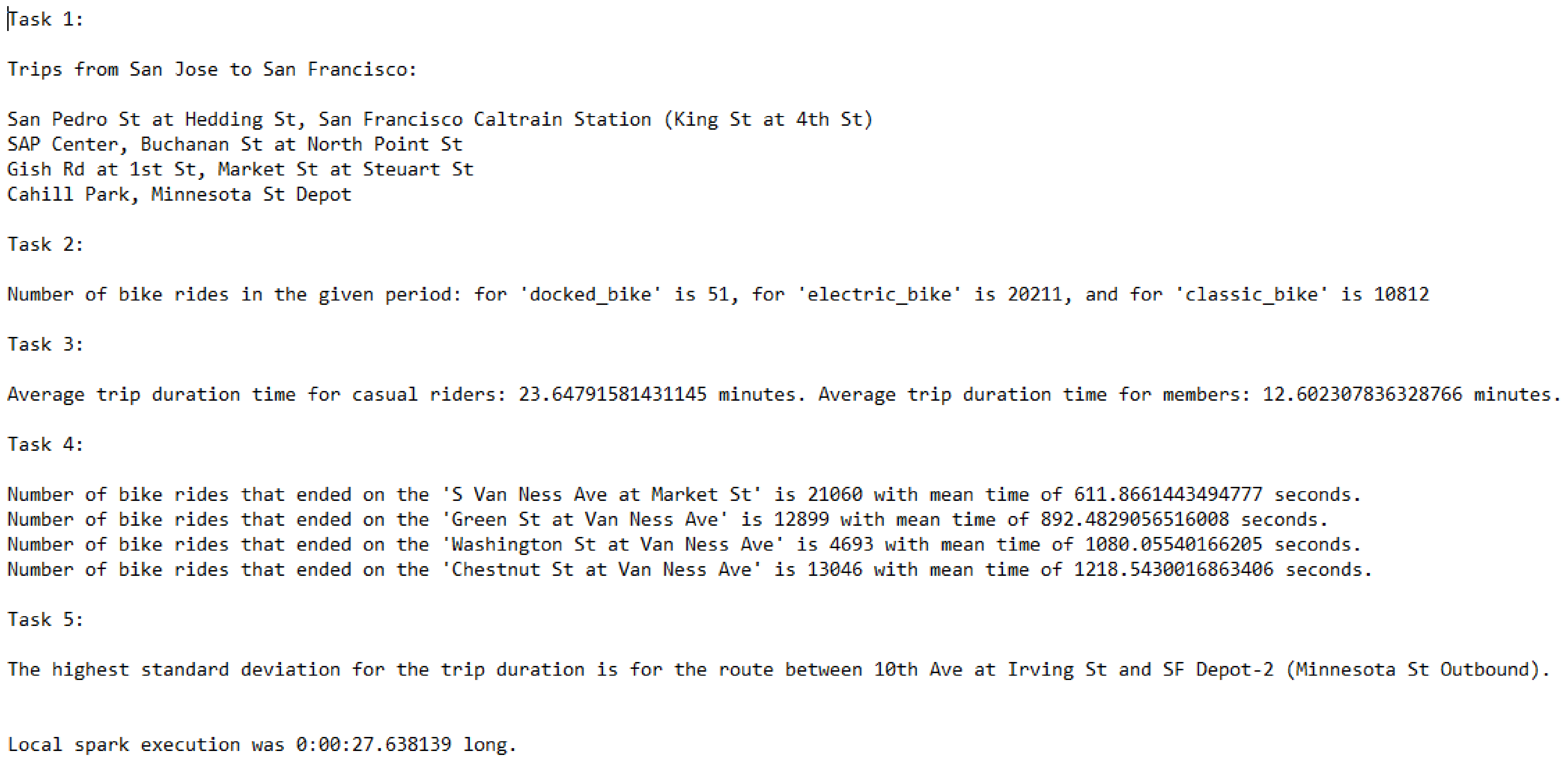
```
# Task 4
task4 = df.groupBy('end_station_name').agg(count('ride_id').alias('ride_count'), mean('duration').alias('mean_trip_time'))\
    .filter(col('end_station_name').like('%Van Ness Ave%'))\
    .sort(col('mean_trip_time').asc()).collect()
output.append('Task 4:\n')
for row in task4:
    task4_result = 'Number of bike rides that ended on the \'' + str(row.asDict()['end_station_name']) + '\' is ' \
        + str(row.asDict()['ride_count']) + ' with mean time of ' + str(row.asDict()['mean_trip_time']) + ' seconds.'
    print(task4_result)
    output.append(task4_result)
```

- Zadatak 5 pronalazi rutu vožnje kod koje je standardna devijacija vremena vožnje bila najviša



```
# Task 5
task5 = df.groupBy('start_station_name', 'end_station_name').agg({'duration': 'stddev'}) \
    .withColumnRenamed('stddev(duration)', 'standard_deviation_time').sort(
    col("standard_deviation_time").desc()).collect()
task5_result = 'The highest standard deviation for the trip duration is for the route between ' \
               + str(task5[0].asDict()['start_station_name']) + ' and ' \
               + str(task5[0].asDict()['end_station_name']) + '.'
print(task5_result, '\n')
output.append('\nTask 5:\n')
output.append(task5_result + '\n')
```

- Rezultat izvršenja aplikacije



```
Task 1:

Trips from San Jose to San Francisco:

San Pedro St at Hedding St, San Francisco Caltrain Station (King St at 4th St)
SAP Center, Buchanan St at North Point St
Gish Rd at 1st St, Market St at Steuart St
Cahill Park, Minnesota St Depot

Task 2:

Number of bike rides in the given period: for 'docked_bike' is 51, for 'electric_bike' is 20211, and for 'classic_bike' is 10812

Task 3:

Average trip duration time for casual riders: 23.64791581431145 minutes. Average trip duration time for members: 12.602307836328766 minutes.

Task 4:

Number of bike rides that ended on the 'S Van Ness Ave at Market St' is 21060 with mean time of 611.8661443494777 seconds.
Number of bike rides that ended on the 'Green St at Van Ness Ave' is 12899 with mean time of 892.4829056516008 seconds.
Number of bike rides that ended on the 'Washington St at Van Ness Ave' is 4693 with mean time of 1080.05540166205 seconds.
Number of bike rides that ended on the 'Chestnut St at Van Ness Ave' is 13046 with mean time of 1218.5430016863406 seconds.

Task 5:

The highest standard deviation for the trip duration is for the route between 10th Ave at Irving St and SF Depot-2 (Minnesota St Outbound).


Local spark execution was 0:00:27.638139 long.
```

Pokretanje aplikacije na kontejnerima imajući izvornu baywheels.csv datoteku sa podacima na HDFS-u

- Preuzeti docker-hadoop repozitorijum sa github-a: https://github.com/big-data-europe/docker-hadoop
- Otvoriti docker-compose.yml datoteku (trenutno sadrži instrukcije za pravljenje sledećih Hadoop servisa: namenode, datanode, resourcemanager, nodemanager1, historyserver)
- Ovoj mreži kontejnera dodati sledeće Spark servise:
  - spark-master:

```
image: bde2020/spark-master:3.1.2-hadoop3.2
    container_name: spark-master
    ports:
      - "8080:8080"
      - "7077:7077"
    environment:
      - INIT_DAEMON_STEP=setup_spark
```

- spark-worker-1:

```
image: bde2020/spark-worker:3.1.2-hadoop3.2
    container_name: spark-worker-1
    depends_on:
      - spark-master
    ports:
      - "8081:8081"
    environment:
      - "SPARK_MASTER=spark://spark-master:7077“
```

- spark-worker-2:

```
image: bde2020/spark-worker:3.1.2-hadoop3.2
    container_name: spark-worker-2
    depends_on:
      - spark-master
    ports:
      - "8082:8081"
    environment:
      - "SPARK_MASTER=spark://spark-master:7077"
```

Finalna verzija docker-compose.yml datoteke je dostupna na sledećem linku: https://github.com/Veljanovskii/bigdata-project-1/blob/main/docker-compose.yml

- Pokrenuti mrežu kontejnera pomoću docker compose up –d komande

- Zatim, potrebno je kopirati datoteku sa izvornim podacima na HDFS
  - Najpre na namenode pomoću:
    - docker cp baywheels.csv namenode:baywheels.csv
  - Pokrenuti namenode pomoću:
    - docker exec –it namenode bash
  - Opciono, napraviti odgovarajući direktorijum gde će datoteka biti smeštena:
    - hdfs dfs –mkdir –p /user/root/input
  - Smestiti izvornu datoteku u dati folder:
    - hdfs dfs –put baywheels.csv /user/root/input/baywheels.csv

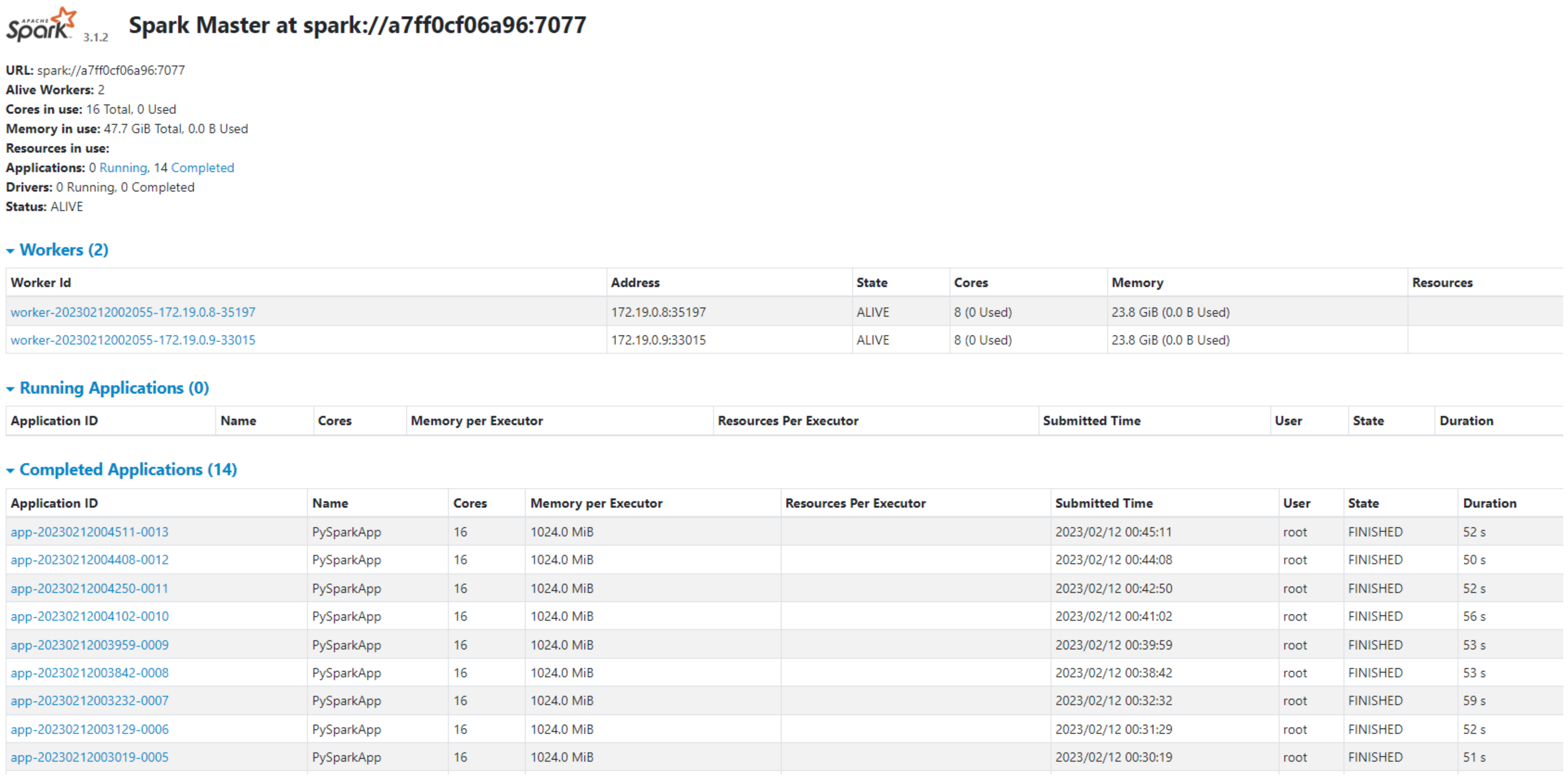
Ovime je csv datoteka podignuta na HDFS i nalazi se na lokaciji: hdfs://namenode:9000/user/root/input/baywheels.csv

- Slično, možemo podići izvorni kod python skripte na spark-app kontejner
  - Spark-app image pravimo na osnovu sledećeg Dockerfile-a komandom docker build --rm -t spark-app .
    - FROM bde2020/spark-python-template:3.1.2-hadoop3.2
    - RUN cd /app pip install -r requirements.txt
    - ENV SPARK_APPLICATION_PYTHON_LOCATION /app/main.py
    - ENV SPARK_APPLICATION_ARGS /app/.env
  - Pokretanje pomoću: docker run --name spark-app --net docker-hadoop_default -p 4040:4040 -d spark-app

- Doker kopiranje:
  - docker cp main.py spark-app:/bin
- Pokrenuti spark-app pomoću:
  - docker exec –it spark-app bash
- Pokrenuti skriptu pomoću:
  - /spark/bin/spark-submit –-master spark://spark-master:7077 main.py

Ovime je aplikacija pokrenuta na klasteru Spark Docker kontejnera

Stanje i tok rada aplikacije se može pratiti na http://localhost:8080/



Spark Master at spark://a7ff0cf06a96:7077

URL: spark://a7ff0cf06a96:7077
Alive Workers: 2
Cores in use: 16 Total, 0 Used
Memory in use: 47.7 GiB Total, 0.0 B Used
Resources in use:
Applications: 0 Running, 14 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers (2)

Worker Id	Address	State	Cores	Memory	Resources
worker-20230212002055-172.19.0.8-35197	172.19.0.8:35197	ALIVE	8 (0 Used)	23.8 GiB (0.0 B Used)	
worker-20230212002055-172.19.0.9-33015	172.19.0.9:33015	ALIVE	8 (0 Used)	23.8 GiB (0.0 B Used)	

Running Applications (0)

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

Completed Applications (14)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20230212004511-0013	PySparkApp	16	1024.0 MiB		2023/02/12 00:45:11	root	FINISHED	52 s
app-20230212004408-0012	PySparkApp	16	1024.0 MiB		2023/02/12 00:44:08	root	FINISHED	50 s
app-20230212004250-0011	PySparkApp	16	1024.0 MiB		2023/02/12 00:42:50	root	FINISHED	52 s
app-20230212004102-0010	PySparkApp	16	1024.0 MiB		2023/02/12 00:41:02	root	FINISHED	56 s
app-20230212003959-0009	PySparkApp	16	1024.0 MiB		2023/02/12 00:39:59	root	FINISHED	53 s
app-20230212003842-0008	PySparkApp	16	1024.0 MiB		2023/02/12 00:38:42	root	FINISHED	53 s
app-20230212003232-0007	PySparkApp	16	1024.0 MiB		2023/02/12 00:32:32	root	FINISHED	59 s
app-20230212003129-0006	PySparkApp	16	1024.0 MiB		2023/02/12 00:31:29	root	FINISHED	52 s
app-20230212003019-0005	PySparkApp	16	1024.0 MiB		2023/02/12 00:30:19	root	FINISHED	51 s