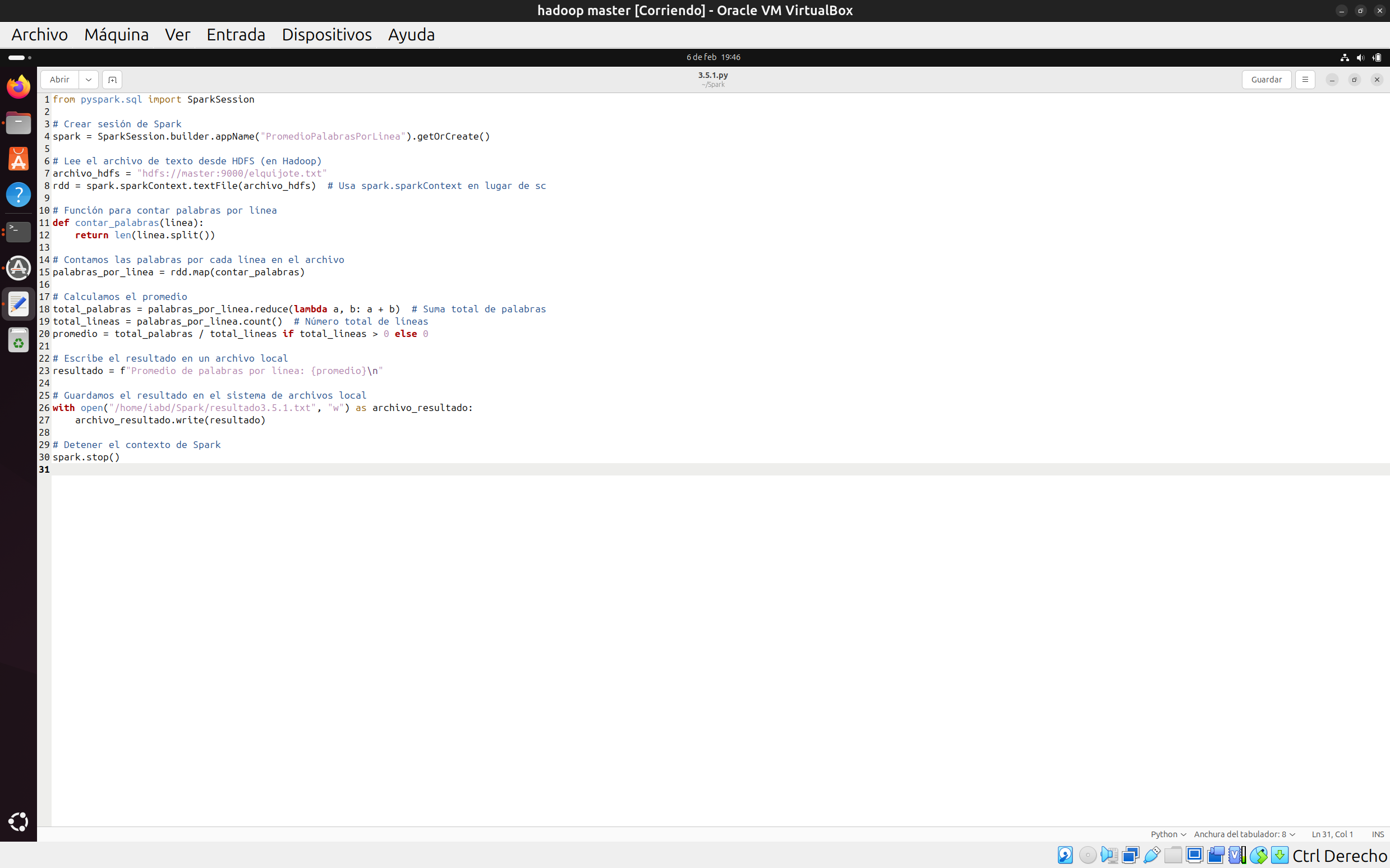
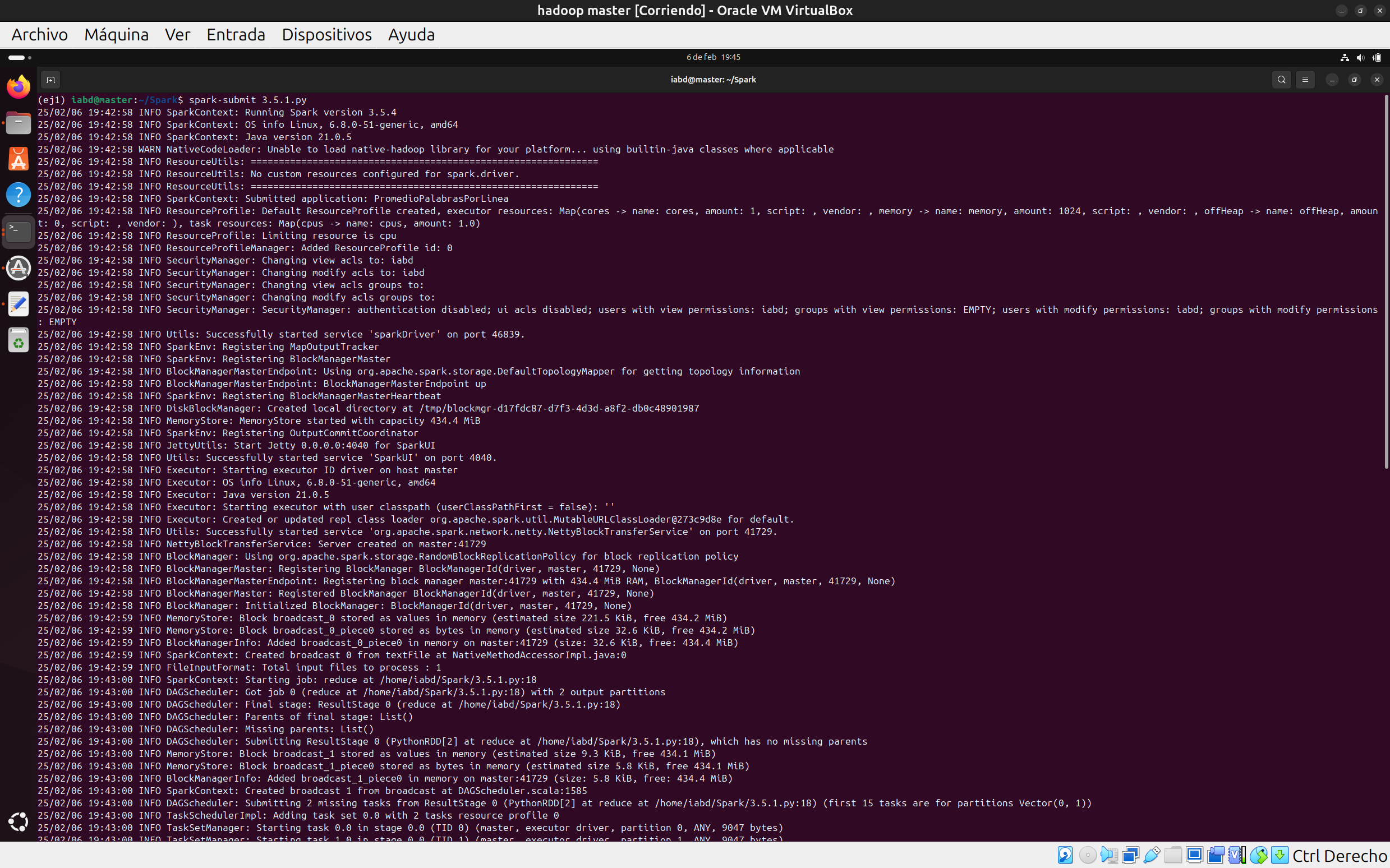
# EJERCICIOS

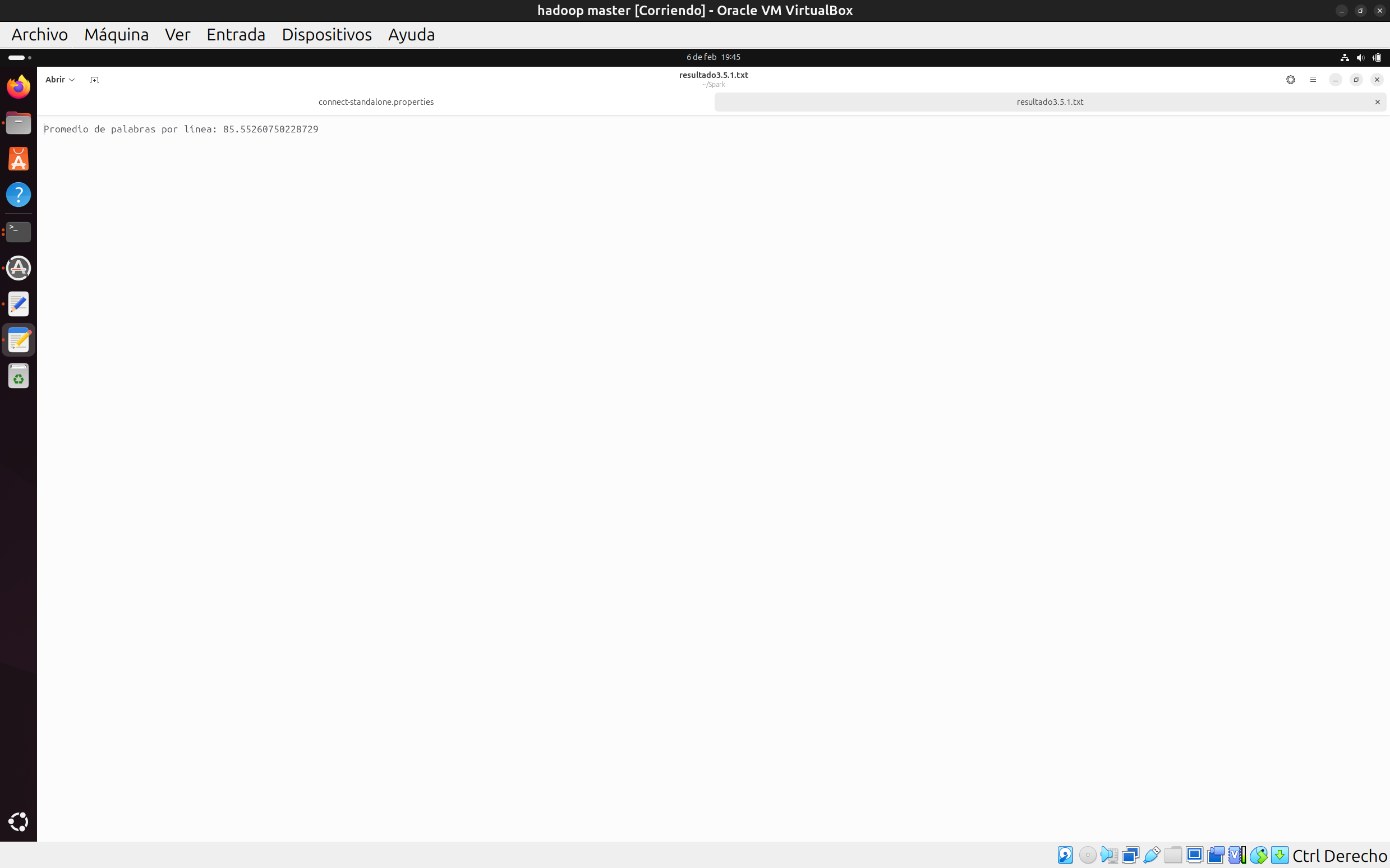
## EJERCICIOS PARA SPARK Y HADOOP EN UBUNTU (USANDO SPARK-SUBMIT)

### EJERCICIO 1

Calcula el número promedio de palabras por línea en un fichero de texto que tengas en Hadoop y guarda el resultado en un fichero (en el sistema operativo, no en Hadoop).

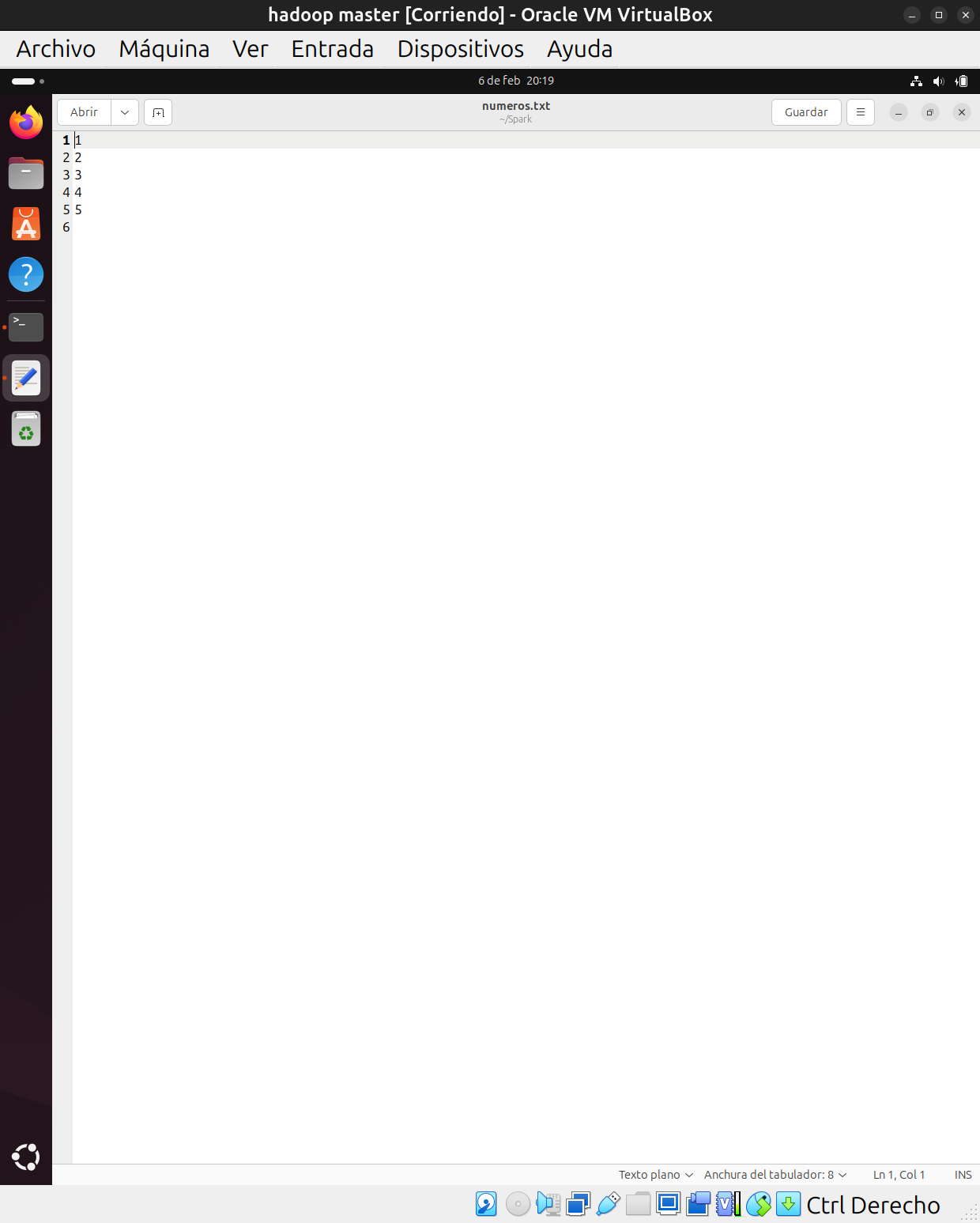


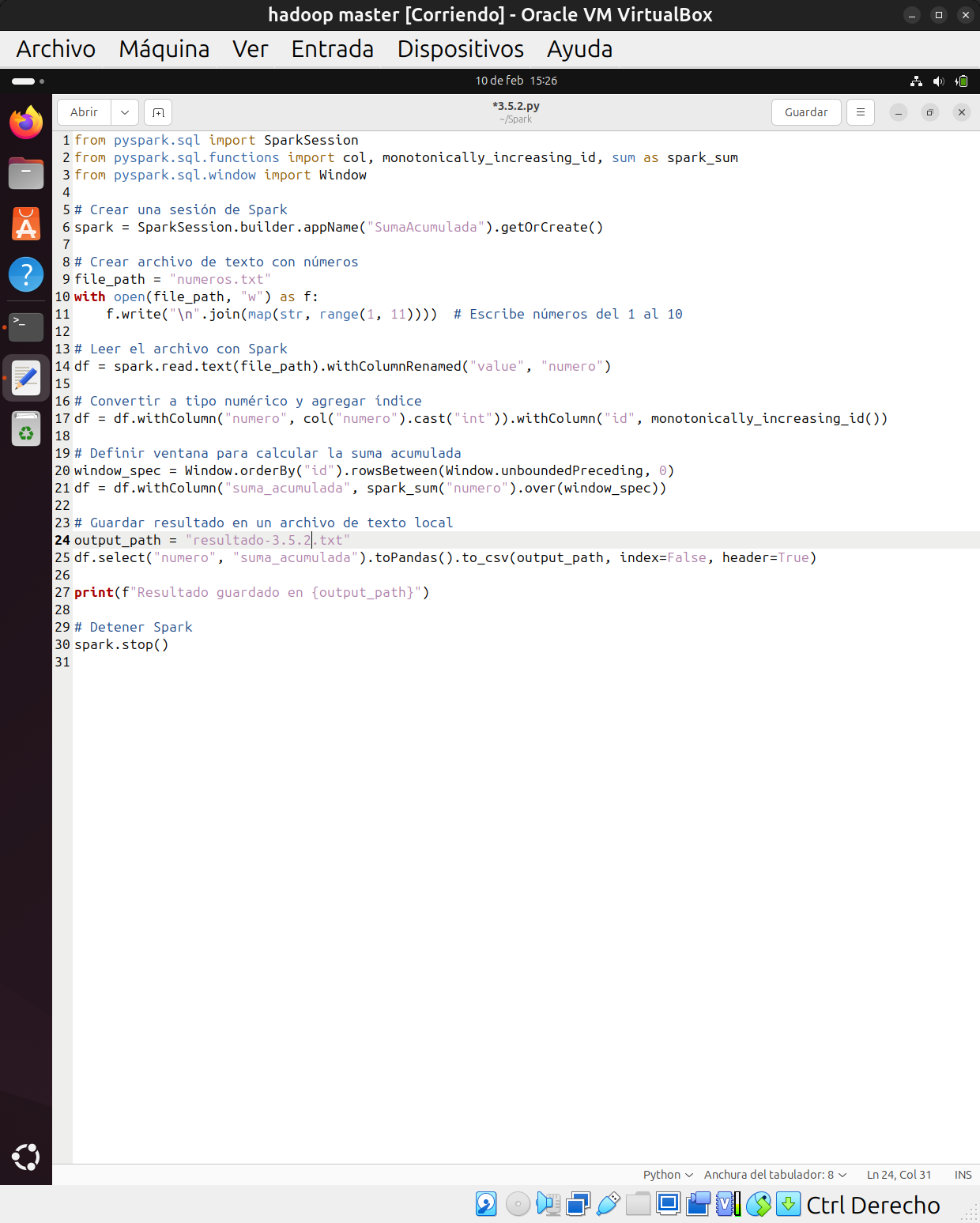


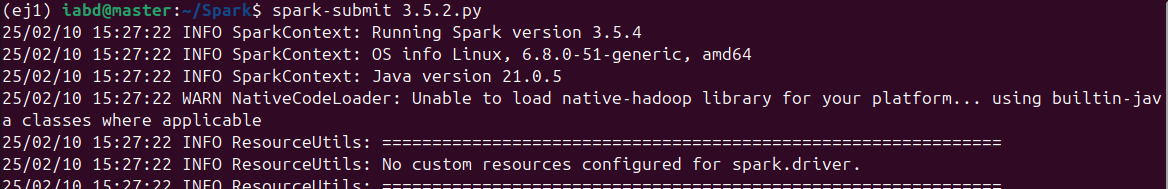


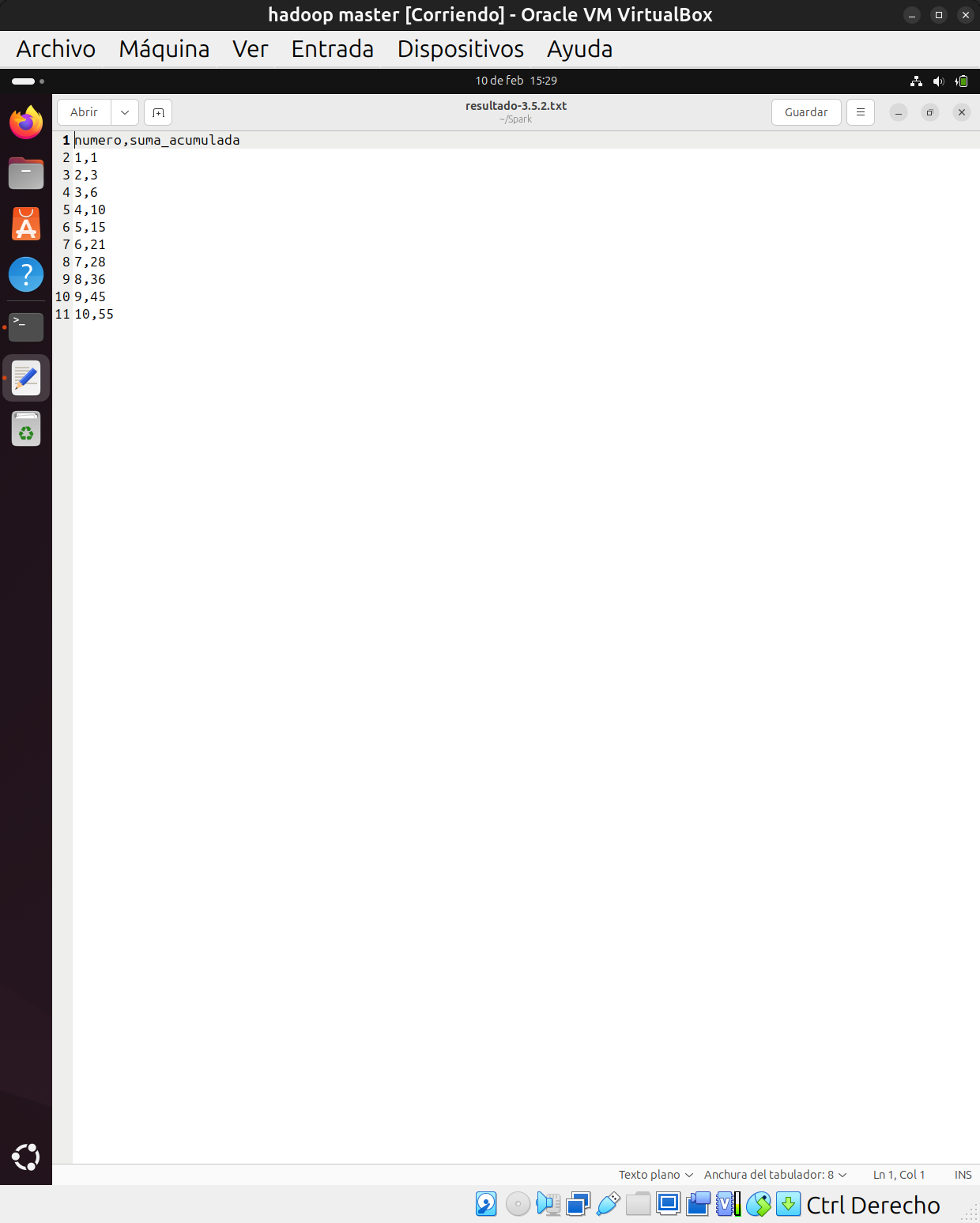
### EJERCICIO 2

Calcula la suma acumulada de números en un archivo de texto (que tiene que estar en local, no en hadoop), donde cada línea contiene un número (tendrás que crear el archivo con los números también). Guarda el resultado en un fichero (en el sistema operativo, no en Hadoop)





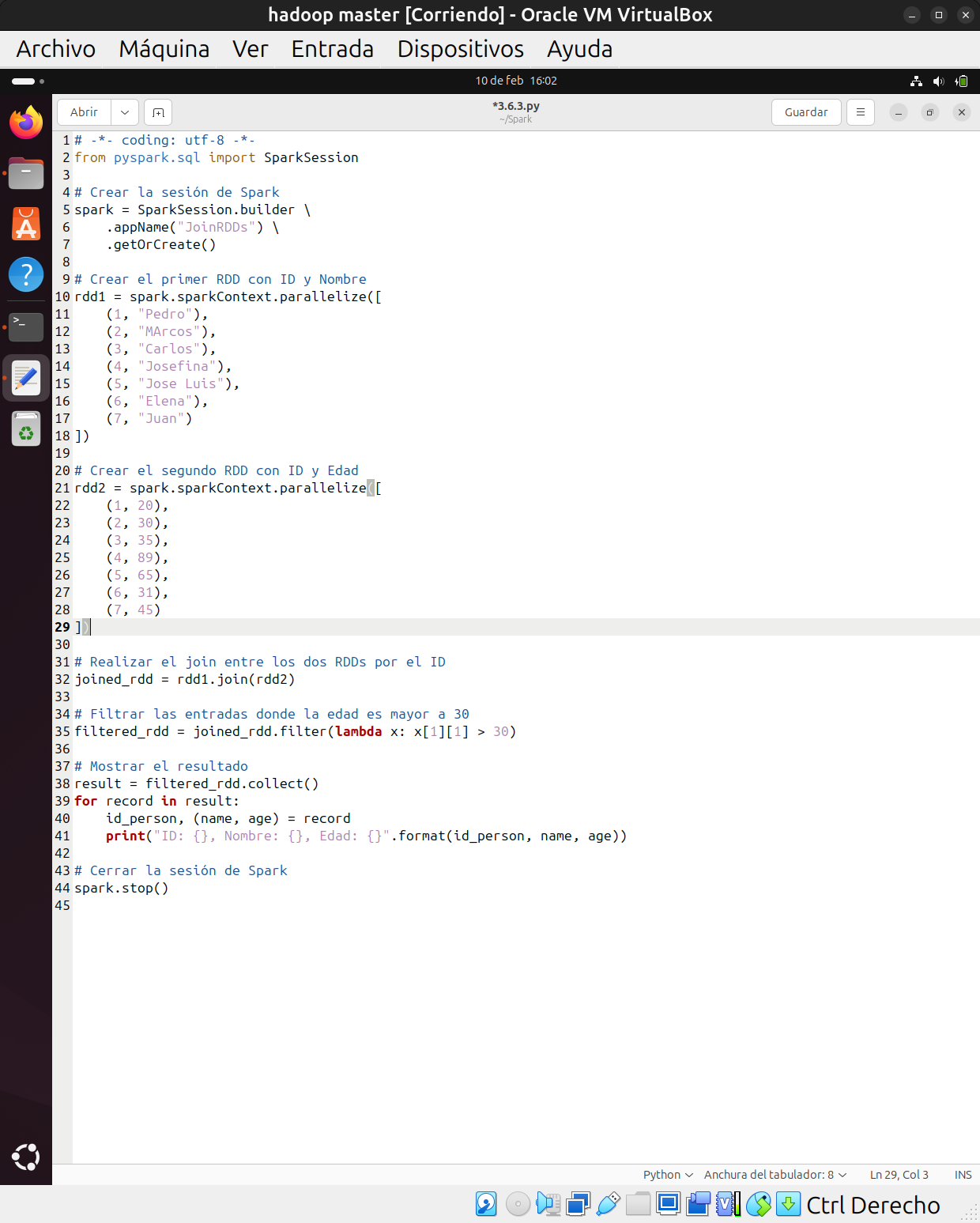


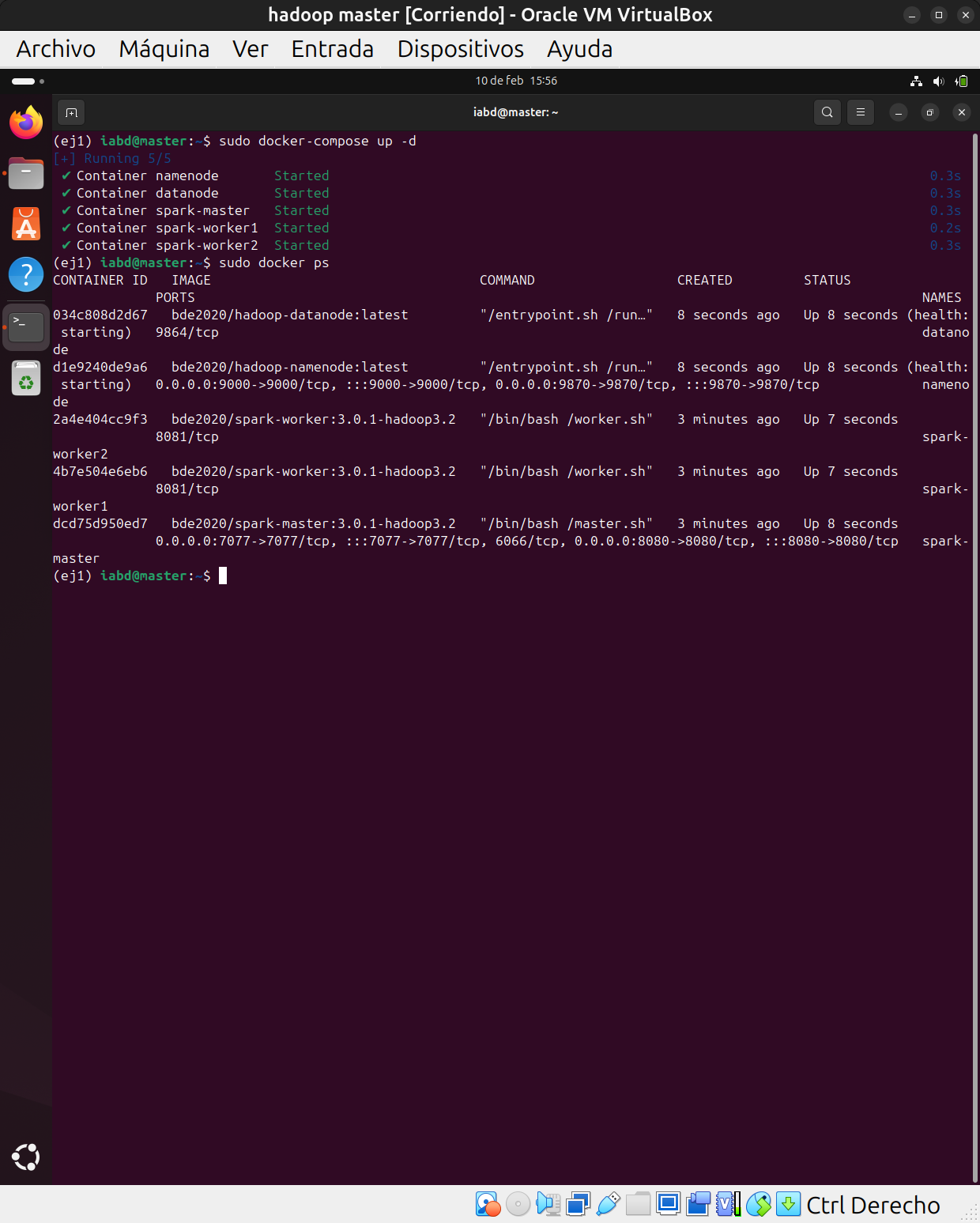


## EJERCICIOS PARA SPARK CON DOCKER EN UBUNTU (USANDO SPARK-SUBMIT)

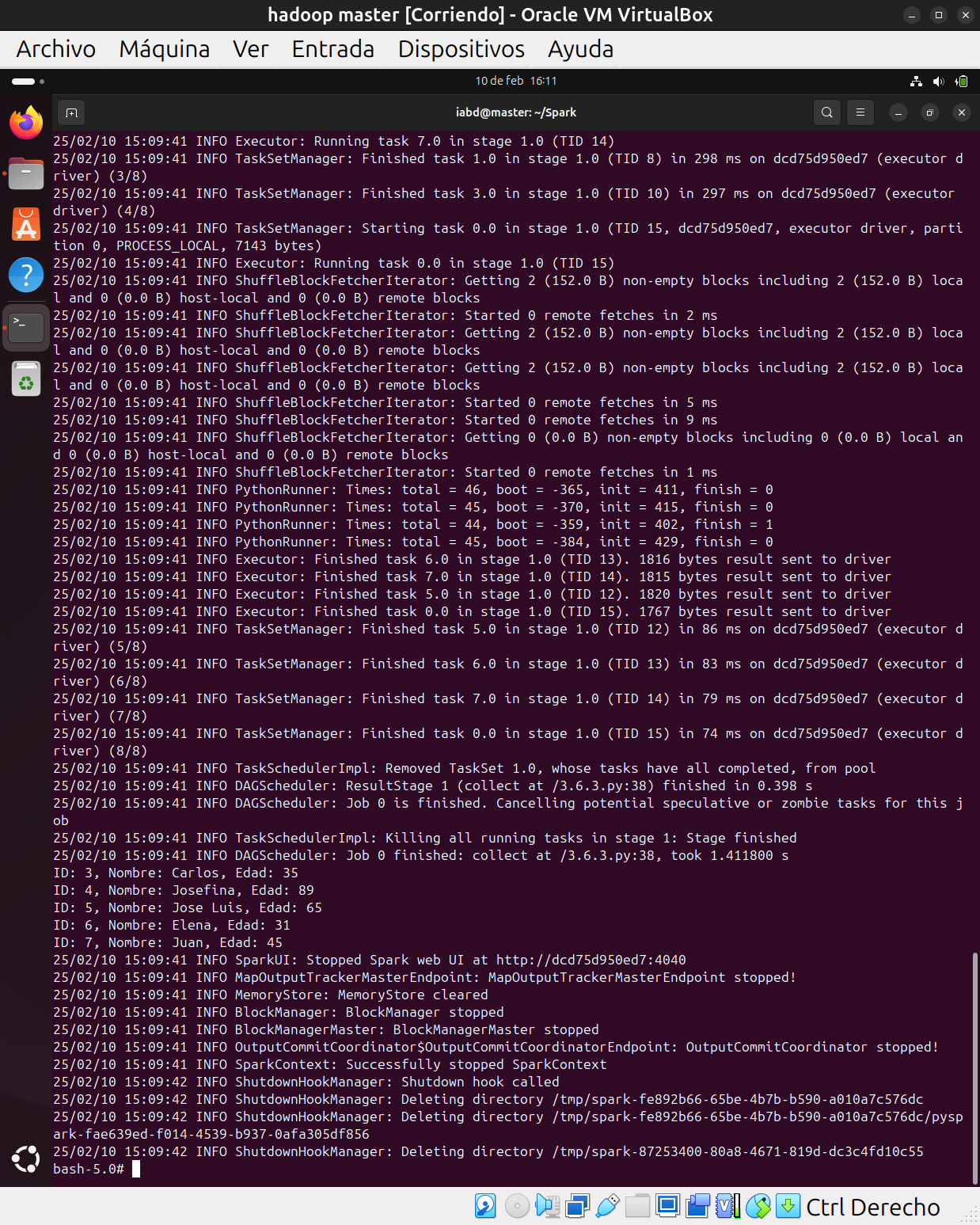
### EJERCICIO 3

Realiza un join entre dos RDDs donde el primero contiene el id y el nombre de varias personas (7 por lo menos) y el segundo el id y la edad y selecciona las entradas donde la edad es mayor a 30.



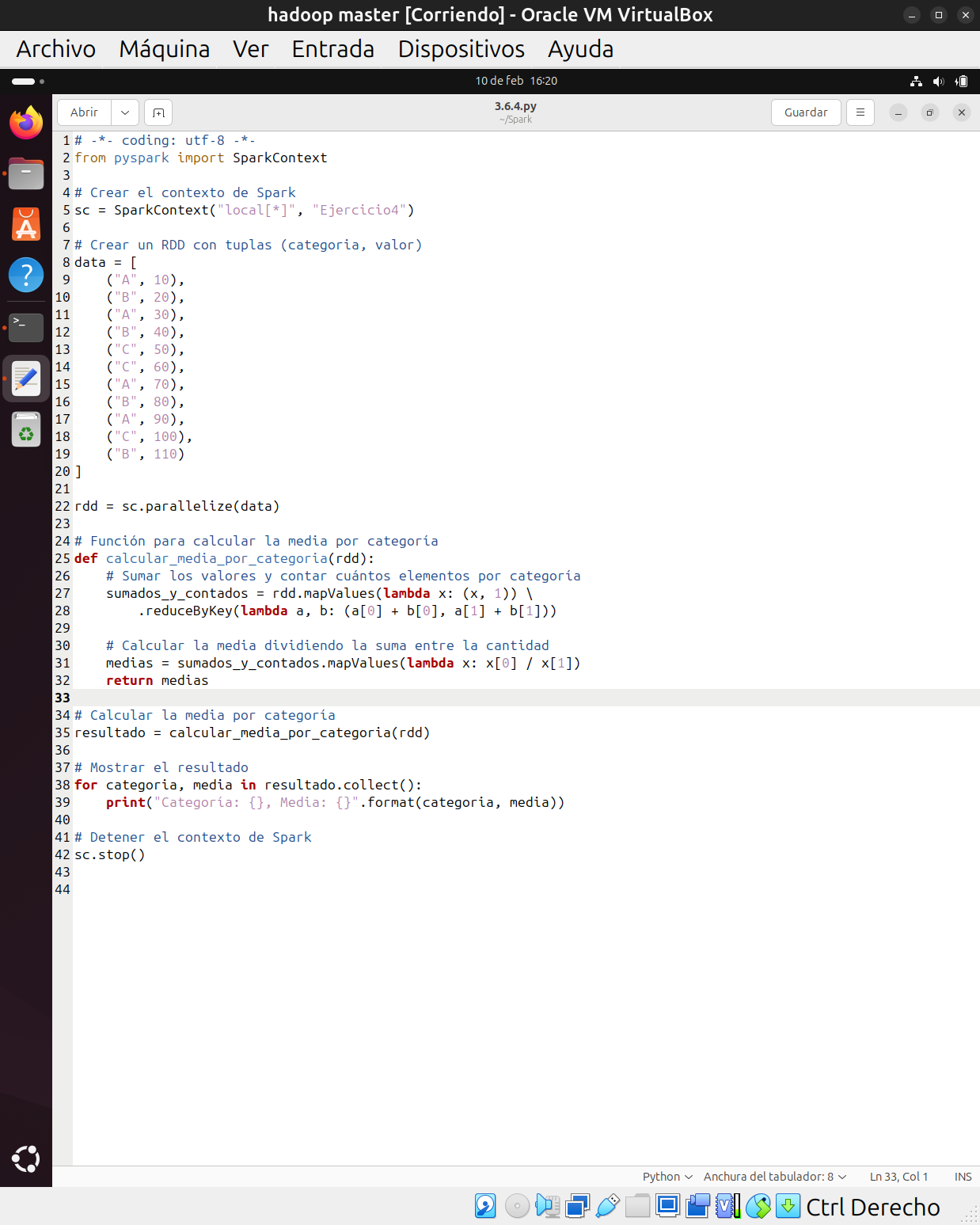


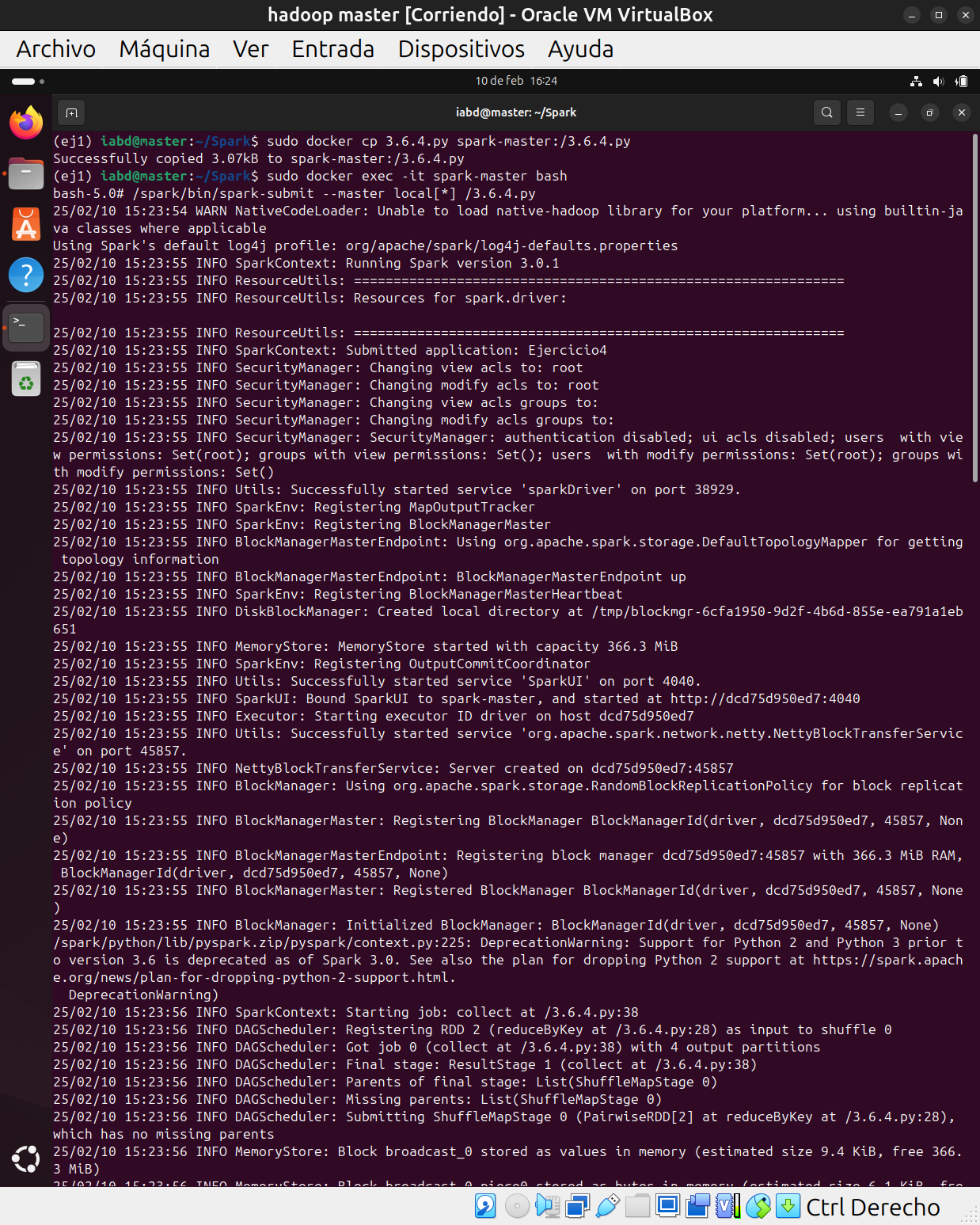


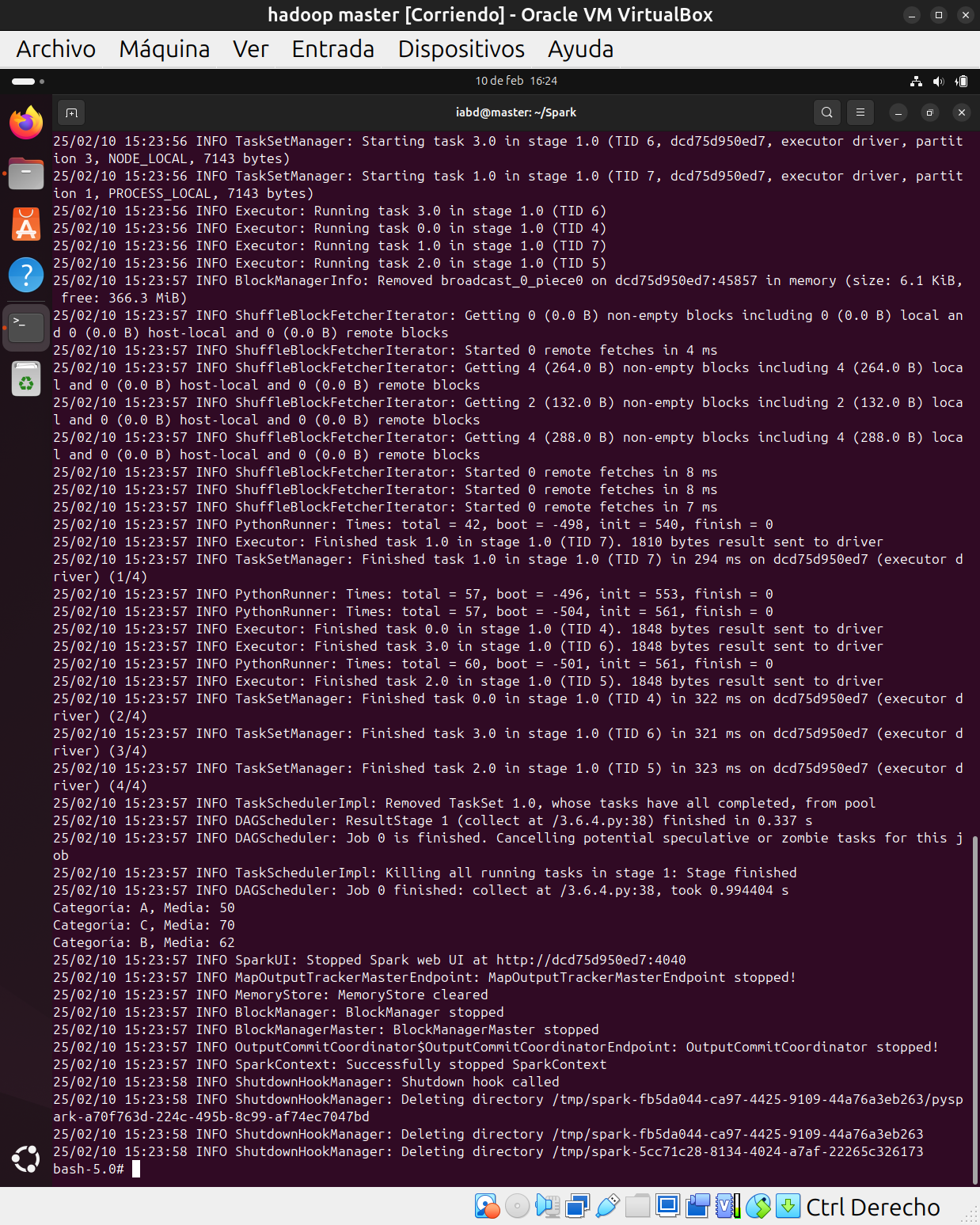


### EJERCICIO 4

Dado un RDD con tuplas (categoria, valor), calcula el valor medio por categoría (tendrás que crear un rdd con al menos 11 líneas).



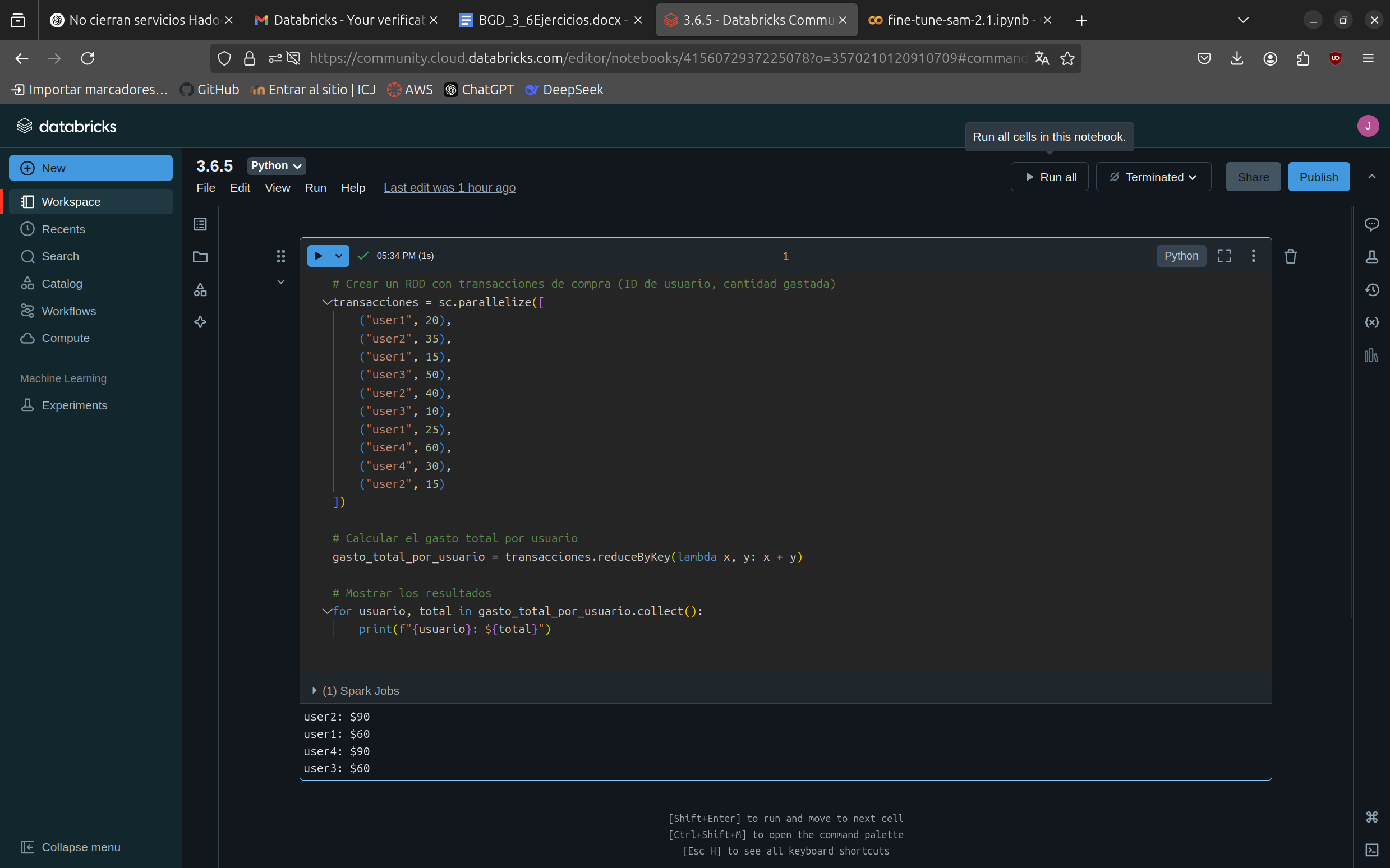




## EJERCICIOS PARA DATABRICKS

### EJERCICIO 5

Utilizando un RDD que contiene transacciones de compra (ID de usuario, cantidad gastada), calcula el gasto total por usuario (el rdd por lo menos tiene que tener 10 líneas).



### EJERCICIO 6

Dado un RDD de valores (timestamp, valor), encuentra la diferencia máxima de valores (el rdd tienes que leerlo desde el archivo times.csv que tendrás que subir a Databricks).

from datetime import datetime

# Leer el archivo CSV en un RDD (asegúrate de que el archivo 'times.csv' esté en el sistema)

rdd = sc.textFile("/FileStore/shared\_uploads/jonfdg@gmail.com/times.csv")

# Eliminar la primera línea (cabecera) y convertir las líneas restantes en tuplas (timestamp, valor)

rdd = rdd.filter(lambda line: not line.startswith("timestamp")).map(lambda line: tuple(line.split(',')))

# Convertir el timestamp de cadena a un valor numérico (epoch timestamp en segundos)

def convertir\_timestamp(timestamp\_str):

# Definir el formato de fecha que se espera (por ejemplo 'YYYY-MM-DD HH:MM:SS')

formato = "%Y-%m-%d %H:%M:%S"

timestamp = datetime.strptime(timestamp\_str, formato)

# Convertir a tiempo Unix (segundos desde el 1 de enero de 1970)

return int(timestamp.timestamp())

# Convertir las líneas a (timestamp, valor) y hacer la conversión del timestamp

rdd = rdd.map(lambda x: (convertir\_timestamp(x[0]), float(x[1])))

# Extraer solo los valores del RDD (sin considerar el timestamp)

valores = rdd.map(lambda x: x[1])

# Encontrar el valor máximo y mínimo

valor\_maximo = valores.max()

valor\_minimo = valores.min()

# Calcular la diferencia máxima

diferencia\_maxima = valor\_maximo - valor\_minimo

# Mostrar el resultado

print(f"Diferencia máxima de valores: {diferencia\_maxima}")



### EJERCICIO 7

Implementa un algoritmo para contar la frecuencia de cada palabra en un conjunto de documentos, muestra la frecuencia de cada palabra ordenada de forma descendente.

from pyspark.sql import SparkSession

import re # Para eliminar signos de puntuación

# Crear o recuperar la sesión de Spark

spark = SparkSession.builder.appName("AnalisisRelatos").getOrCreate()

# Leer los archivos en la carpeta 'relatos'

ruta\_relatos = "/FileStore/shared\_uploads/jonfdg@gmail.com/relatos/\*"

rdd = spark.sparkContext.textFile(ruta\_relatos)

# Función para limpiar y dividir las líneas en palabras

def limpiar\_y\_dividir(linea):

# Eliminar signos de puntuación y convertir a minúsculas

linea\_limpia = re.sub(r'[^\w\s]', '', linea.lower())

# Dividir en palabras

return linea\_limpia.split()

# Procesar el RDD

rdd\_palabras = rdd.flatMap(limpiar\_y\_dividir) # Dividir en palabras

frecuencia\_palabras = rdd\_palabras.map(lambda palabra: (palabra, 1)) \

.reduceByKey(lambda x, y: x + y) # Contar frecuencia

# Ordenar palabras por frecuencia (descendente)

palabras\_ordenadas = frecuencia\_palabras.sortBy(lambda x: x[1], ascending=False)

# Recolectar y mostrar resultados

resultados = palabras\_ordenadas.collect() # Recolectar los resultados

for palabra, frecuencia in resultados:

print(f"{palabra}: {frecuencia}")

