Pyspark week 11
================

1. cache & persist

2. spark-submit

```
from sys import stdin

sc = SparkContext("local[*]","PremiumCustomers")

base_rdd = sc.textFile("/Users/trendytech/Desktop/data/customer-orders.csv")

mapped_input = base_rdd.map(lambda x: (x.split(",")[0],float(x.split(",")[2])))

total_by_customer = mapped_input.reduceByKey(lambda x,y: x+y)

premium_customers = total_by_customer.filter(lambda x: x[1] > 5000)

doubled_amount = premium_customers.map(lambda
x:(x[0],x[1]*2)).persist(StorageLevel.MEMORY_ONLY)

result = doubled_amount.collect()

for x in result:
   print(x)

print(doubled_amount.count())

stdin.readline()
```

```
spark-submit /Users/trendytech/PycharmProjects/pysparklearning/module1.py
```

Ratings.dat
Movies.dat

find the top rated movies

1. atleast 1000 people have rated
2. rating should be > 4.5

finding top movies (scala code)
===================================

```
import org.apache.log4j.Logger
import org.apache.log4j.Level
import org.apache.spark.SparkContext


object JoinDemo extends App {


Logger.getLogger("org").setLevel(Level.ERROR)
val sc = new SparkContext("local[*]","joindemo")

val ratingsRdd= sc.textFile("/Users/trendytech/Desktop/data/ratings.dat")

 val mappedRdd = ratingsRdd.map(x => {
   val fields = x.split("::")
   (fields(1),fields(2))
 })

 val newmappedRdd= mappedRdd.mapValues(x=>(x.toFloat,1.0))

 val reduceRdd = newmappedRdd.reduceByKey((x,y) => (x._1+y._1, x._2+y._2))

 val filteredRdd = reduceRdd.filter(x=>x._2._1 > 1000)

 val finalRdd = filteredRdd.mapValues(x => x._1/x._2).filter(x => x._2 > 4.5)

 val moviesRdd= sc.textFile("/Users/trendytech/Desktop/data/movies.dat")

 val moviesmappedRdd = moviesRdd.map(x => {
   val fields = x.split("::")
   (fields(0),(fields(1),fields(2)))
 })

 val joinedRdd = moviesmappedRdd.join(finalRdd)

 val topMoviesRdd = joinedRdd.map(x=>x._2._1)

 topMoviesRdd.collect.foreach(println)
}
```

=========================
equivalent pyspark code
=========================

```
from pyspark import SparkContext

sc = SparkContext("local[*]","joindemo")

ratings_rdd = sc.textFile("/Users/trendytech/Desktop/data/ratings.dat")

mapped_rdd = ratings_rdd.map(lambda x: (x.split("::")[1], x.split("::")[2]))

new_mapped_rdd = mapped_rdd.mapValues(lambda x: (float(x),1.0))

reduce_rdd = new_mapped_rdd.reduceByKey(lambda x,y: (x[0]+y[0], x[1]+y[1]))

filtered_rdd = reduce_rdd.filter(lambda x: x[1][0] > 1000)

final_rdd = filtered_rdd.mapValues(lambda x: x[0]/x[1]).filter(lambda x: x[1] > 4.5)

movies_rdd= sc.textFile("/Users/trendytech/Desktop/data/movies.dat")

movies_mapped_rdd = movies_rdd.map(lambda x: (x.split("::")[0],(x.split("::")[1],x.split("::")[2])))

joined_rdd = movies_mapped_rdd.join(final_rdd)

top_movies_rdd = joined_rdd.map(lambda x: x[1][0])

result = top_movies_rdd.collect()

for x in result:
   print(x)
```

Structured API's
==================

DataFrame, DataSets, Spark SQL

DataSets are not supported.

SparkSession

```
from pyspark import SparkConf
from pyspark.sql import SparkSession

my_conf = SparkConf()
my_conf.set("spark.app.name", "my first application")
my_conf.set("spark.master","local[*]")

spark = SparkSession.builder.config(conf=my_conf).getOrCreate()

orderDf = spark.read.csv("/Users/trendytech/Desktop/data/orders.csv")

orderDf.show()

spark.stop()
```

=======

find the total orders placed by each customer where customer id > 10000

```
from pyspark import SparkConf
from pyspark.sql import SparkSession

my_conf = SparkConf()
my_conf.set("spark.app.name", "my first application")
my_conf.set("spark.master","local[*]")


spark = SparkSession.builder.config(conf=my_conf).getOrCreate()

orderDf =
spark.read.option("header",True).option("inferSchema",True).csv("/Users/trendytech/Desktop/data/orders.csv")

groupedDf = orderDf.repartition(4) \
.where("order_customer_id > 10000") \
.select("order_id","order_customer_id") \
.groupBy("order_customer_id") \
.count()

groupedDf.show()
```

===============

1. wrong column name

so when we give a column name which does not exist then the error is shown at runtime and not at compile time.

2. standard way

```
orderDf = spark.read.format("csv")\
    .option("header",True)\
    .option("inferSchema",True)\
    .option("path","/Users/trendytech/Desktop/data/orders.csv")\
    .load()

orderDf = spark.read.format("json")\
    .option("path","/Users/trendytech/Desktop/data/orders.json")\
    .load()
```