# Esercizio 1

All’inizio, per generare le finestre dell’1%, ho pensato a una soluzione basata su RDD, dato che tramite la sua ripartizione avrei avuto la possibilità di dividere facilmente il dataset nel modo desiderato e poi effettuare le operazioni dovute. Ho però riscontrato alcune criticità in questo approccio:

1. L’algoritmo tendeva a creare risultati diversi da quelli previsti, dato che durante la ripartizione dell’RDD non viene conservato l’ordine dei dati;
2. L’RDD, rispetto al DataFrame, non presenta una componente di ottimizzazione della computazione. Ciò porta ad un deterioramento percepibile delle perfomance, con un tempo di esecuzione di che si aggira tra i 45-60 secondi su 10 milioni di parametri;
3. In ultima battuta, il fatto che il numero di worker(1) sia molto minore rispetto al numero di partizioni dell’RDD(100), non dà la possibilità di sfruttare al meglio la ripartizione stessa dato che alla fine i task verranno eseguiti sempre dallo stesso worker.

Per ovviare a questo ho creato quindi una seconda soluzione al problema, basandomi sui DataFrame offerti da Spark. Per ovviare al problema della partizione, ho definito una colonna di appoggio con degli indici che vanno da [1, numero di partizioni(ES:100)]. Questi indici sono assegnati come delle finestre, quindi nel caso della prima finestra, i primi centomila elementi avranno indice 1 e così via. In questo modo ho potuto effettuare il calcolo della somma dei valori di una finestra in modo più efficiente, potendo sfruttare al meglio Spark. Per completare, ho calcolato la somma cumulata creando una nuova colonna e, per terminare l’esercizio, ho eliminato la colonna che conteneva il valore della somma dei valori interni alle singole finestre dato che non è richiesto nell’output.

Nel caso in cui i dati fossero molto più grandi, potremmo avere un problema di performance dato che rischiamo di saturare la memoria del singolo esecutore che stiamo usando. Una soluzione potrebbe essere quella di utilizzare diversi esecutori in parallelo, in modo da suddividere il carico su diversi esecutori piuttosto che su uno singolo.

# Esercizio 2

Data la natura più strutturata del dato rispetto al primo esercizio, ho deciso di utilizzare direttamente i DataFrame per effettuare il job. Inizialmente andiamo quindi a calcolare la media dei valori della colonna Value e l’andiamo a materializzare con il comando *head()* e successivamente andiamo ad aggregare sulla colonna ID e andiamo a calcolare la media dei valori della colonna Value per ogni ID associato. Per poter calcolare quindi la distanza, andiamo prima a creare una colonna con all’interno che è il risultato della differenza tra la media di ogni gruppo e la media totale della colonna Value. Andando a visionare i valori generati c’erano alcuni problemi di underflow e per risolverli sono andato ad effettuare un’operazione di arrotondamento dei valori.

Per quanto riguarda questo esercizio non mi vengono in mente particolari criticità se non quella definita precedentemente anche per l’esercizio 1.