Corso di Ingegneria Informatica - Big Data
A.A. 2021/2022

Progetto di contabilità - R e MapReduce su Hadoop

Nome dello studente:	*Antonino Giuseppe Giano*
Matricola	*672HHHINGINFOR*
Data Appello	*Luglio 2022*
Esame	*Introduzione ai Big Data*

Indice

Introduzione

La presente relazione, in risposta all'esercizio sulla contabilità del corso di Introduzione ai Big Data, andrà a valutare alcuni aspetti contabili della ditta STECCAPARAPETUTTI S.R.L.. Il testo dell'esercizio si trova in Appendice A.

Le tecnologie coinvolte per lo sviluppo in **R** sono:

- RStudio 2022.02.2+485
- R 4.2.0 Patched

R è un linguaggio di programmazione per l'analisi statistica e la produzione di grafici. Può essere scaricato dal sito ufficiale.

Per programmare in R si è scelto di utilizzare **RStudio Desktop**, un IDE open source, anch'esso scaricabile dal sito ufficiale.

Per l'utilizzo del paradigma **MapReduce** si è usato:

- Ubuntu 20.04
- Oracle JDK 8
- Hadoop 3.2.1
- R 4.2.0 Patched

IL SO Ubuntu è stato installato su una macchina virtuale creata con Hyper-V 10.0.19041, disponibile nativamente su Windows 10 Pro. Si è scelto questo sistema operativo perché open source, affidabile e di semplice installazione Alla macchina virtuale sono stati concessi:

- RAM: dinamica (minima 512 MB, massima 8 GB)
- processori virtuali: 4
- disco: 40GB

Hadoop è una piattaforma che permette di eseguire elaborazioni su grandi quantità di dati distribuendo su cluster per velocizzare e rendere meno dispendiosa l'esecuzione. Questo è possibile grazie al paradigma MapReduce che individua due fasi principali, quella di **mapping** e di **reducing**, i cui i dati vengono:

- divisi in **split** per essere inviati ai **Data Node**, dove viene applicato loro il task di **mapping**, il quale produce dei risultati che vengono distribuiti a vari Data Node (questa operazione è detta **shuffling**);
- si applica il task di **reducing** all'output del mapping, ottenendo così il risultato finale

Inoltra Hadoop mette a disposizione **HDFS** (Hadoop Distributed File System). Esso permette di gestire grandi dataset su cluster di hardware commerciale.

Svolgimento

Di seguito vengono riportati i job da realizzare per una lettura più agevole:

1. Calcola la media di vendita per ogni mese di ogni anno
2. Calcola la varianza di vendita per ogni mese di ogni anno
3. Identifica il mese di ogni anno con maggiore vendita
4. Identifica il mese di ogni anno con minore vendita

Per semplicità, i job sono stati prima programmati in R. I risultati ottenuti in questa fase sono stati poi confrontati con quelli ottenuti con il paradigma MapReduce.

Analisi preliminare

Il file **Ordini.csv**, reso disponibile sulla piattaforma Uninettuno, rappresenta il file di input. Aprendo il file su un qualsiasi editor di testo si capisce la struttura delle righe, che possiedono tre campi. Il primo (d'ora in poi **Tipo**) assume diversi valori. Estrapolando questi valori con R, otteniamo (l'ordine non è di nessuna rilevanza):

```
> orders = read.csv(file = "./ordini.csv", header = F, sep = ",")
> unique(orders[, 1])
[1] "FATTURA"          "RICEVUTA"         "NOTA.DI.CREDITO" "DDT"                "OFFERTA"
[6] "PREVENTIVO"       "INVENTARIO"       "BUONO.PRELIEVO"
```

Figura 1 – Valori assunti dalla prima campo del file Ordini.csv

Si hanno dunque 8 tipologie di ordini. Dopo un approfondimento (Appendice B) al riguardo, le tipologie di interesse, cioè quelle assimilabili a "vendita", sono: **FATTURA**, **NOTA.DI.CREDITO**.
Il secondo campo è la **data** in formato **YYYYMMDD**.
Il terzo campo è l'**importo** in Euro relativo all'ordine.

Vale la pena soffermarsi su quanto richiesto. Il primo job e il secondo job chiedono di analizzare mese per mese l'andamento delle vendite. Il terzo e il quarto job possono essere interpretati in almeno tre modi:

1. identificare, per ogni anno, i mesi con la singola vendita di minore e maggiore valore durante l'arco dello stesso anno;
2. identificare, per ogni anno, i mesi con il numero minore e maggiore di vendite;
3. identificare, per ogni anno, i mesi in cui l'importo totale delle vendite è minore e maggiore.

La prima possibilità non sembra avere nessuna utilità pratica, mentre la seconda potrebbe essere interessante per una efficiente gestione del magazzino o degli ordini ai fornitori.
La terza alternativa è stata implementata nel progetto perché quest'opzione è quella che porta maggior valore nel contesto di un'analisi poco approfondita delle vendita di una società.

Sviluppo in R

Innanzitutto, le istruzioni *jobs.r - da 1 a 3* cambiano la directory di riferimento (da cui vengono calcolati i path relativi) con quella in cui jobs.r è locato. Questo semplifica le variabili che descrivono i path che si trovano nelle istruzioni *jobs.r - da 6 a 11*.
Inoltre, viene caricato il package **dplyr**[1] utilizzato per le sue funzioni, ma principalmente per l'operatore **pipe** che rende `x %>% f(y)` equivalente a `f(x, y)`, il che evita di dover salvare i risultati intermedi e di ottenere un codice altamente leggibile.
Viene controllato che le directory da usare esistano e, in caso contrario, le si creano.

Successivamente viene letto l'input, inserendolo nel data frame *orders*. Come si intuisce in *jobs.r - 23*, il file Ordini.csv è sprovvisto di riga di intestazione. Quindi, in *jobs.r - 20* vengono aggiunti i nomi delle colonne al data frame.
In *jobs.r - da 21 a 23* viene filtrato il data frame, conservando le sole righe di interesse per i job nel data frame *sales*.

Per implementare i job 1 e 2 si usa la funzione *aggregate()*[2] in *jobs.r - 33 e 41*. Questa funzione raggruppa i dati passati come primo parametro in base alla clausola specificata nel parametro *by*. Ad ognuno di questi gruppi viene poi applicata la funzione descritta nel parametro *FUN* (*mean* per la media e *var* per la varianza).
La funzione *var* restituisce la **varianza campionaria** $\frac{\sum_{i=1}^{n}(x_i-\bar{x})_.}{n-1}$.Volendo ottenere la **varianza della popolazione** $\frac{\sum_{i=1}^{n}(x_i-\bar{x})_.}{n-1}$, si moltiplica per *(n-1)/n* a riga *jobs.r - 42 e 44*.
Il risultato ottenuto viene inserito in un data frame, il quale viene utilizzato per generare il file CSV di output con la funzione *write.csv()*.

Lo sviluppo dei job 3 e 4 è stato quasi esclusivamente portato a termine grazie a *dplyr*. Viene preventivamente creato *sum_by_month*, un data frame che contiene la somma degli importi delle vendite per ogni mese di ogni anno. A questo data frame viene aggiunta una colonna, *Year*, contenente l'anno della vendita.
Dunque, la ricerca dei valori minimi e massimi (*jobs.r - da 58 a 60 e da 68 a 70*) avviene raggruppando le righe del data frame appena creato per *Year* e ricercando, per ogni gruppo, una volta la somma minima e una volta la somma massima.
Anche per i job 3 e 4 i risultati vengono inseriti in un data frame che vengono scritti in output su file CSV.

Per brevità, si limita la discussione riguardo i grafici. Si vuole solo far notare che per i jobs 1 e 2 si sono prodotti dei diagrammi a barre mensili. Per i job 3 e 4, invece, è stato prodotto un solo grafico che contiene sia i valori minimi, sia i valori massimi.
Sono stati generati altri due grafici:

- **Monthly sales distribution over the years**, spiega quanto differiscono le vendite mensili per ogni anno; sono state scelte due rappresentazioni per mostrare come uno stesso concetto può essere visualizzato in modo più o meno accattivante;

- **Monthly sales over the years**, sovrappone le vendite mensili di ogni anno (escluso il 2020 per mancanza di dati); contiene due regressioni: una lineare, che mostra una tendenza costante delle vendite tra inizio e fine di ogni anno, e una LOESS, che rivela dei picchi nelle vendite.

MapReduce con Hadoop e R

Per l'installazione di Hadoop su Ubuntu è stata seguita una guida online[3].
Fatti i dovuti test, è iniziata la fase di sviluppo degli script R per i mapper e i reducer.
Inizialmente, è stata creata la directory */prj* sul filesystem distribuito con il comando:
`hdfs dfs -mkdir /prj`

Per ogni job sono stati realizzati due script in R, uno per la fase di mapping e uno per la fase di reducing, e uno script shell per il setup del filesystem distribuito e l'esecuzione di mapper e reducer. L'unico vera differenza tra i job risiede negli script reducer, in quanto la fase di mapping è uguale per tutti e per gli script l'unica differenza è nei percorsi di file e cartelle.

Lo script shell non è assolutamente necessario per lo scopo del progetto, ma ha reso più semplici le fasi di sviluppo, debug e stesura della relazione. Inizialmente, esso elimina dal filesystem distribuito la cartella *prj/j1* (o del job che si vuole eseguire) e la ricrea per essere sicuri che non vi siano sottocartelle all'interno. In realtà, si vuole eliminare la cartella dove viene scritto l'output (che è *prj/j1/output*), perciò basterebbe cancellare solamente quest'ultima.
Successivamente viene copiato il file di input sul filesystem distribuito (*j1/cmd.sh - 7*) e vengono modificati i permessi di accesso ai file *mapper.r* e *reducer.r* garantendo privilegi massimi.
A questo punto, viene eseguito il job tramite il comando *hadoop jar* a cui viene passato il percorso del jar di hadoop, il mapper e il reducer in R, il file di input e la cartella di output.
Infine, viene mostrato a schermo l'output prodotto.

Il mapper, comune a tutti job, dichiara due funzioni, *trim_white_space* e *split_into_fields*, che servono rispettivamente a rimuovere eventuali spazi all'inizio e alla fine di una stringa e a recuperare i valori da una stringa in formato CSV. Inoltre, viene dichiarato un array di stringhe che contiene i tipi di documenti di interesse per il progetto (*j1/mapper.r - 4*).
Viene poi aperta una connessione per la lettura della console (*j1/mapper.r - 7*). Questo permette di legge il file passato come input nel comando *hadoop jar*. Attraverso un ciclo while si legge riga per riga il file. Ogni riga viene pulita, separata in campi e stampata in output, se rientra nei tipi ammessi. Infine viene chiusa la connessione con la console.

Il reducer del job 1, anch'esso, dichiara la funzione *trim_white_space*. In più, dichiara *split_line*, una funzione per ottenere una lista, che rappresenta la singola vendita, a partire da una stringa.
In *j1/reducer.r - 9* viene creato un **environment** che permette di associare dei valori a delle chiavi, le quali sono delle stringhe. In questo caso, le chiavi sono i mesi e i valori sono una lista fatta da un contatore e il totale delle vendite per quel mese.
La lettura dell'input, del tutto analoga a quella di *j1/mapper.r*, questa volta restituisce per

intero quanto è stato prodotto dal mapper. Si procede iterando le righe lette, pulendo ogni riga ed estrapolandone le informazioni, cioè mese e importo della vendita.
Per ogni vendita, si controlla se il mese è già presente nell'ambiente *env*. In caso negativo, si associa al mese una lista con *count* uguale a 1 e *tot* pari all'importo della vendita. Altrimenti, invece, si conserva l'attuale valore associato al mese, per poi modificarlo incrementando di un'unità il contatore e aggiungendo a *tot* l'importo della vendita di quella interazione.
Finito il ciclo, si chiude la connessione con la console.
Infine, si iterano tutte le chiavi dell'ambiente *env*, cioè i mesi, se ne estrapola la media e si stampa in output il risultato.

Lo script del reducer per il job 2 conta poche differenze rispetto a quello per il job 1.
In primis, viene utilizzato il package *data.table* per leggere dal filesystem distribuito le medie calcolate dal job 1 (*j2/reducer.r - 14*). Infine, l'elemento *tot* degli oggetti in *env* contiene la somma degli scarti al quadrato, anziché la somma degli importi come nel job 1. Questo valore viene calcolato con una funzione ad hoc, *sqm*.

Per ultimo, si analizza il reducer per il job 3, in quanto il reducer per il job 4 è del tutto identico se non per l'applicazione della funzione *which.min* al posto di *which.max*.
Anche questo reducer dichiara le due funzioni *trim_white_space* e *split_line*, inizializza un environment e legge l'input di cui si iterano le righe. Una grande differenza sta nell'associazione del valore alla chiave nell'environment. Similmente a quanto accade nel job 1, viene calcolata la somma degli importi delle vendite per ogni mese, ma lo si è fatto con una diversa sintassi, la quale risulta più leggibile. Il reducer continua copiando tutti i valori nell'environment dentro un dataframe, *values*. Infine, si itera attraverso gli anni delle vendite e per ogni anno si filtra *values* affinchè contenga solo le righe relative a quell'anno. Si estrae la riga che contiene il valore massimo (o minimo nel caso del job 3) e la si stampa in output.

Conclusioni

Nelle prime fasi del progetto è stato di fondamentale importanza l'esplorazione dei dati e del loro significato, sia con l'uso di R, sia attraverso ricerche su internet (si pensi ai tipi di documenti del dataset).

Si vuole anche far presente come con R si sia potuto dare risposta, in meno di 100 righe di codice, a delle richieste, seppur semplici, che sono;

1. Calcola la media di vendita per ogni mese di ogni anno
2. Calcola la varianza di vendita per ogni mese di ogni anno
3. Identifica il mese di ogni anno con maggiore vendita
4. Identifica il mese di ogni anno con minore vendita

Inoltre, si è voluto dare un assaggio della potenza espressiva di R generando dei grafici altamente personalizzati.

Spostando l'attenzione sullo sviluppo attraverso il paradigma Map Reduce, ci si è trovati davanti all'insufficienza di risorse hardware per avviare un cluster multinodo.
Inoltre, non sono state usate sandbox preconfezionate, in quanto si ritiene altamente formativo configurare ambienti di sviluppo che vengono utilizzati in contesti lavorativi.
Quindi, per avere accesso a un cluster multinodo, la scelta sarebbe dovuta ricadere sull'installazione di diverse virtual machine sull'unico pc a disposizione. Avendo dovuto scartare questa alternativa, si è scelto di creare un clutere single-node allo scopo di vedere in azione il YARN e Hadoop, pur sapendo che questa non rappresenta la migliore strada per ottenere delle performance ottimali in contesti Big Data.

Possibili sviluppi futuri

L'analisi condotta non tiene conto delle note di credito perché non le si può correlare temporalmente (cosa necessaria, visto che i dati sono raggruppati mese per mese). Avendo a disposizioni tali dati, di cui si è evidenziata la mancanza in Appendice B, si potrebbero correggere i risultati ottenuti. Inoltre, si potrebbe enfatizzare l'analisi delle note di credito per capire quanto spesso queste modificano gli importi della fatturazione, in quali periodi dell'anno.

Altri benefici si potrebbero ottenere confrontando i dati ottenuti con dei dati analoghi riferiti a competitor diretti di mercato o nuove società che si configurano come possibili rivali. Ovviamente per questo tipo di indagine servirebbe un data con maggiore varietà di dati come, ad esempio, informazioni sui clienti/acquirenti, dimensione dell'azienda, ecc…

Avendo a disposizione più dati si potrebbe ricostruire in modo migliore la storia della società. Per di più, se si stesse parlando di una società che produce beni non personalizzabili, si potrebbe prevedere l'andamento delle vendite future attraverso un modello di Machine Learning, il che permetterebbe di gestire meglio problemi logistici o di produzione.

Riferimenti bibliografici, Sitografia

[1] Introduction to dplyr

[2] aggregate: Compute Summary Statistics of Data Subsets

[3] How to Install Hadoop on Ubuntu 18.04 or 20.04

[b1] Fattura - Wikipedia

[b2] Documenti fiscali — gdf.gov.it

[b3] Note di variazione Iva: termini e modalità di emissione - Fiscomania

[b4] Note di variazione Iva: termini e modalità di emissione - Fiscomania

[b5] Documento di trasporto - Wikipedia

[b6] Offerta pubblica di acquisto - Wikipedia

[b7] Libro degli inventari: cos'è ea cosa serve? - Fiscomania.

Appendice A: Testo dell'esercizio

CORSO INTRODUZIONE AI BIG DATA

La ditta STECCAPARAPETUTTI S.R.L. ha un sistema di contabilità che memorizza tutti gli ordini fatti (per ordini si intendono Fatture, Buoni Prelievo, Offerte…). In previsione dell'anno successivo, la ditta vorrebbe sapere il suo andamento di vendita.

Scaricato il file **Ordini.csv** strutturato in questa maniera:

```
-----------------------------------------------------
Tipo-documento,data(aaaammgg),costo(€)
-----------------------------------------------------
FATTURA, 20160104,139.8
FATTURA, 20160104,169.2
FATTURA, 20160104,65.3
……
```

Progettare e realizzare per ogni anno i seguenti job:

1. Calcola la media di vendita per ogni mese di ogni anno
2. Calcola la varianza di vendita per ogni mese di ogni anno
3. Identifica il mese di ogni anno con maggiore vendita
4. Identifica il mese di ogni anno con minore vendita

Per ciascun job bisogna illustrare e documentare in un rapporto finale:

- Implementazione R
- Implementazione del paradigma mapReduce di HADOOP.
  Nel caso in cui si verificassero problemi con l'installazione e il funzionamento di HADOOP con R allora si può utilizzare il TOOL.

Deve essere consegnato solamente il rapporto finale scritto come una relazione. Con i codici allegati.
I Risultati ottenuti devono essere illustrati tramite dei grafici.

Suggerimento: i documenti tipo "offerta" sono da considerarsi come una vendita effettiva?

Appendice B: Tipologie di ordini

Fattura: la fattura è un documento che formalizza un credito (fattura di vendita) o un debito (fattura di acquisto)[b1]. Nel file Ordini.csv non vengono date indicazioni sul tipo di fatture registrate nel sistema. Per Dunque, si è scelto di considerare i record di tipo FATTURA come fatture di vendita, che quindi verranno prese in considerazione durante lo svolgimento del progetto.

Ricevuta: alcuni soggetti sono esonerati dall'obbligo di emissione della fattura, se non espressamente richiesto dal cliente.
Nei casi in cui non venga emessa la fattura, il corrispettivo dei beni o servizi ceduti viene certificata attraverso lo scontrino o la ricevuta fiscale[b2].
Si considera dunque la ricevuta come una "vendita".

Nota di credito: si è supposto che con NOTE.DI.CREDITO il sistema registri uno dei tipi di "Variazioni dell'imponibile o dell'imposta" descritte nell'articolo 26 del DPR 633/72[b3]. Queste servono a correggere l'imponibile o l'imposta di una fattura precedentemente emessa.
Nello specifico, la nota di credito (o di accredito) è una nota di variazione in diminuzione, serve quindi a diminuire il debito del cliente verso il soggetto che rilascia la nota[b4]. Questa potrebbe essere considerata una vendita (in negativo), in quanto va a modificare l'importo della fattura associata alla vendita.
Il file Ordini.csv, non dà modo di risalire alla fattura modificata dalla singola nota di credito. Quindi, non è possibile decurtare l'importo della nota dall'importo della fattura, questo anche perché la nota può essere emessa anche a mesi di distanza dall'emissione della fattura. Dunque le NOTE.DI.CREDITO non vengono prese in considerazione.

Documento di trasporto: è un documento che viene prodotto per la movimentazione di merci.[b5]
Non costituisce una vendita.

Offerta: si è inteso OFFERTA come Offerta Pubblica di Acquisto[b6], la quale non è una vendita.

Preventivo: il preventivo viene usato per fare un calcolo approssimativo e preventivo dei costi per un servizio. Non è una vendita.

Inventario: si è inteso INVENTARIO come Libro degli Inventari, un documento che riepiloga la situazione economica dell'impresa[b7]. Non è una vendita.

Buono prelievo: sono dei documenti che facilitano il prelievo di materiali o contanti. Non è una vendita.

Appendice C: Codici sorgente e risultati

Codice jobs R

jobs.r

```
1.  #setting working directory and loading libraries
2.  current_working_dir <- dirname(rstudioapi::getActiveDocumentContext()$path)
3.  setwd(current_working_dir)
4.  library(dplyr)
5.
6.  #defining some useful variables
7.  input <- "./ordini.csv"
8.  out_j1 <- "./output/j1.csv"
9.  out_j2 <- "./output/j2.csv"
10. out_j3 <- "./output/j3.csv"
11. out_j4 <- "./output/j4.csv"
12. month_names <- c("Gen", "Feb", "Mar", "Apr", "Mag", "Giu",
13.                  "Lug", "Ago", "Set", "Ott", "Nov", "Dic")
14.
15. if (!dir.exists("./output")) {
16.   dir.create("./output")
17. }
18. if (!dir.exists("./plots")) {
19.   dir.create("./plots")
20. }
21.
22. # reading input
23. orders <- read.csv(file = input, header = F, sep = ",")
24. # adding column names
25. colnames(orders) <- c("Type", "Date", "Euro")
26. # filtering
27. filter_type <- c("FATTURA", "RICEVUTA")
28. sales <- orders[which(orders$Type %in% filter_type), ]
29. # getting all the dates in format "YYYYMM"
30. months <- substring(sales$Date, 1, 6)
31.
32. # JOB 1: calculating the mean values of each month
33. mean_by_month <- aggregate(sales$Euro, by = list(months), FUN = mean)
34. df_mean_by_month <- data.frame(
35.   month = c(paste(substring(mean_by_month[, 1], 5, 6), "-",
36.                   substring(mean_by_month[, 1], 1, 4), sep = "")),
37.   value = round(mean_by_month[, 2], 2))
38. write.csv(df_mean_by_month, out_j1, row.names = F)
39.
40. # JOB 2: calculating the var values of each month
41. variance_by_month <- aggregate(sales$Euro, by = list(months), FUN = var)
42. # 'var' function calculate the sample variance, so the calculated variance
43. # is multiplied by (n-1)/n to obtain population variance
44. n <- aggregate(sales$Euro, by = list(months), FUN = length)
45. variance_by_month$x <- variance_by_month$x * (n$x - 1) / n$x
46. df_variance_by_month <- data.frame(
47.   month = c(paste(substring(variance_by_month[, 1], 5, 6), "-",
48.                   substring(variance_by_month[, 1], 1, 4), sep = "")),
49.   value =  as.numeric(variance_by_month[, 2]))
50. write.csv(df_variance_by_month, out_j2, row.names = F)
51.
52. # calculating the sum of the sale amounts of each month
53. sum_by_month <- aggregate(sales$Euro, by = list(months), FUN = sum)
```

```
# adding the column Year to sum_by_month
sum_by_month$Year <- substring(sum_by_month$Group.1, 1, 4)

# JOB 3: found the most profitable month of each year
max_month_by_year <- sum_by_month %>%
  group_by(Year) %>%
  slice(which.max(x))
df_max_month_by_year <- data.frame(
  month = c(paste(substring(max_month_by_year$Group.1, 5, 6), "-",
                  max_month_by_year$Year, sep = "")),
  value =  as.numeric(max_month_by_year$x))
write.csv(df_max_month_by_year, out_j3, row.names = F)

# JOB 4: found the least profitable month of each year
min_month_by_year <- sum_by_month %>%
  group_by(Year) %>%
  slice(which.min(x))
df_min_month_by_year <- data.frame(
  month = c(paste(substring(min_month_by_year$Group.1, 5, 6), "-",
                  min_month_by_year$Year, sep = "")),
  value =  as.numeric(min_month_by_year$x))
write.csv(df_min_month_by_year, out_j4, row.names = F)

library(ggplot2)
library(patchwork)
#function for styling axes
custom_axes <-
  function(d) {
    return(d +
             theme(axis.text = element_text(size = 15),
                   axis.title = element_text(size = 20, face = "bold"),
                   plot.title = element_text(size = 20, face = "bold")))
  }

# extra (not required)
# let's have a look of the distribution of the monthly sale
# amounts over the years (no 2020 because it's not a full year)
png("./plots/monthly_distributions_of_each_year.png", 1600, 800)
g1 <- ggplot(sum_by_month, aes(x = substr(Group.1, 1, 4), y = x,
                               color = substr(Group.1, 1, 4))) +
  ggtitle("Monthly sales distribution over the years") +
  labs(x = "Date", y = "Euro") +
  scale_color_brewer(palette = "Dark2", guide = "none") +
  theme(panel.background = element_rect(
    fill = "white", color = "black", size = 1),
    panel.grid.major = element_line(
        color = "gray10", size = .5, linetype = "dashed"
    ),
    panel.grid.minor = element_line(
      color = "gray70", size = .25, linetype = "dashed"
    )
   )

 g1 <- custom_axes(g1)
 ((g1 + geom_boxplot(size = 1)) +
     (g1 + geom_violin(fill = "gray80", size = 1.2, alpha = .5) +
        geom_point(size = 3)))
 dev.off()

 # extra (not required)
```

```
114.  # let's have a look of the differences among the years
115.  png("./plots/month_sales_over_years.png", 900, 900)
116.  g2 <-
117.    ggplot(sum_by_month[-which(sum_by_month$Year == "2020"), ],
118.           aes(x = as.numeric(substr(Group.1, 5, 6)),
119.               y = x,
120.               color = Year)) +
121.    ggtitle("Monthly sales over the years") +
122.    labs(x = "Month", y = "Sales Amount") +
123.    scale_color_brewer(palette = "Dark2") +
124.    theme(panel.background = element_rect(
125.      fill = "white", color = "black", size = 1),
126.      panel.grid.major = element_line(
127.        color = "gray10", size = .5, linetype = "dashed"
128.      ),
129.      panel.grid.minor = element_line(
130.        color = "gray70", size = .25, linetype = "dashed"
131.      )
132.    )
133.  g2 <- custom_axes(g2)
134.  g2 + geom_point(size = 2) + geom_line(aes(color = Year)) +
135.    stat_smooth(method = "lm", se = F,
136.                color = "black", size = 1.1) +
137.    stat_smooth(method = "loess", se = F,
138.                color = "red", size = 1.1) +
139.    scale_x_discrete(limits = 1:12,
140.                     breaks = 1:12,
141.                     labels = month_names)
142.  dev.off()
143.
144.  # defining function to plot some similar diagrams
145.  custom_diagram <-
146.    function(df, file_name, file_width = 1600, file_height = 800,
147.             d_title = "Title", d_x_title = "X", d_y_title = "Y",
148.             use_labels = F) {
149.      png(file_name, file_width, file_height)
150.
151.      l <- length(df$month)
152.      diag <- ggplot(df, aes(x = as.numeric(substr(month, 1, 2)), y =
    value)) +
153.        labs(x = d_x_title, y = d_y_title) +
154.        ggtitle(d_title) +
155.        theme_minimal() +
156.        scale_x_discrete(limits = 1:l,
157.                         breaks = 1:l,
158.                         labels = month_names[1:l]) +
159.        geom_bar(stat = "identity",
160.                 color = "black", fill = "blue") +
161.        scale_y_continuous(labels = scales::number)
162.
163.      if (use_labels) {
164.        diag <- diag +
165.          geom_text(aes(label = value),
166.                    color = "black", size = 8, vjust = -.3,
167.                    position = position_dodge(.9))
168.      }
169.      diag <- custom_axes(diag)
170.      print(diag)
171.      dev.off()
172.    }
```

```
173.
174.  # generating diagrams for mean values and variance of each month
175.  for (yearly_means in
176.       split(df_mean_by_month, substring(df_mean_by_month$month, 4, 7))) {
177.    year <- substring(yearly_means[1, 1], 4, 7)
178.    custom_diagram(df = yearly_means,
179.                   file_name = paste("./plots/month_means_", year, ".png"
180.                                     , sep =  ""),
181.                   d_title = paste("Monthly Mean Sales Amount - ", year
182.                                   , sep =  ""),
183.                   d_x_title = "Month",
184.                   d_y_title = "Euro",
185.                   file_width = 1200,
186.                   use_labels = T)
187.  }
188.  for (yearly_vars in
189.       split(df_variance_by_month, substring(df_variance_by_month$month, 4,
    7))) {
190.    year <- substring(yearly_vars[1, 1], 4, 7)
191.    custom_diagram(df = yearly_vars,
192.                   file_name = paste("./plots/month_vars_", year, ".png"
193.                                     , sep =  ""),
194.                   d_title = paste("Monthly Sales Variance - ", year
195.                                   , sep =  ""),
196.                   d_x_title = "Month",
197.                   d_y_title = "Variance amount",
198.                   file_width = 1200,
199.                   use_labels = F)
200.  }
201.
202.  # generating diagrams for the most and the lest profitable month of each
    year
203.  df_min_month_by_year$type <- "min"
204.  df_max_month_by_year$type <- "MAX"
205.  df_min_max <- rbind(df_min_month_by_year, df_max_month_by_year)
206.  df_min_max$Year <- substring(df_min_max$month, 4, 7)
207.  df_min_max <- df_min_max[with(df_min_max, order(Year, value)), ]
208.  png("./plots/min_max.png", 1200, 900)
209.
210.  diag <- ggplot(df_min_max, aes(x = Year, y = value, fill = type)) +
211.    labs(x = "Month", y = "Euro") +
212.    ggtitle("The most and the lest profitable month of each year") +
213.    theme(panel.background = element_rect(
214.      fill = "white", color = "black", size = 1),
215.      panel.grid.major = element_line(
216.        color = "gray10", size = .5, linetype = "dashed"
217.      ),
218.      panel.grid.minor = element_line(
219.        color = "gray70", size = .25, linetype = "dashed"
220.      ),
221.      legend.position = "none"
222.    )
223.  diag <- custom_axes(diag)
224.  diag + geom_bar(position = position_dodge(), stat = "identity",
225.                  color = "black", size = .5) +
226.    scale_fill_manual(values = c("limegreen", "orangered")) +
227.    geom_text(aes(label  = month_names[as.numeric(substr(month, 1, 2))]),
228.              vjust = 1.1, color = "black", size = 5,
229.              position = position_dodge(.9)) +
230.    geom_text(aes(label = value, vjust = ifelse(type == "min", -.3, 2.5)),
```

```
231.              color = "black", size = 5,
232.              position = position_dodge(.9))
233.
234.  dev.off()
235.
236.  # closing all graphics devices
237.  graphics.off()
```

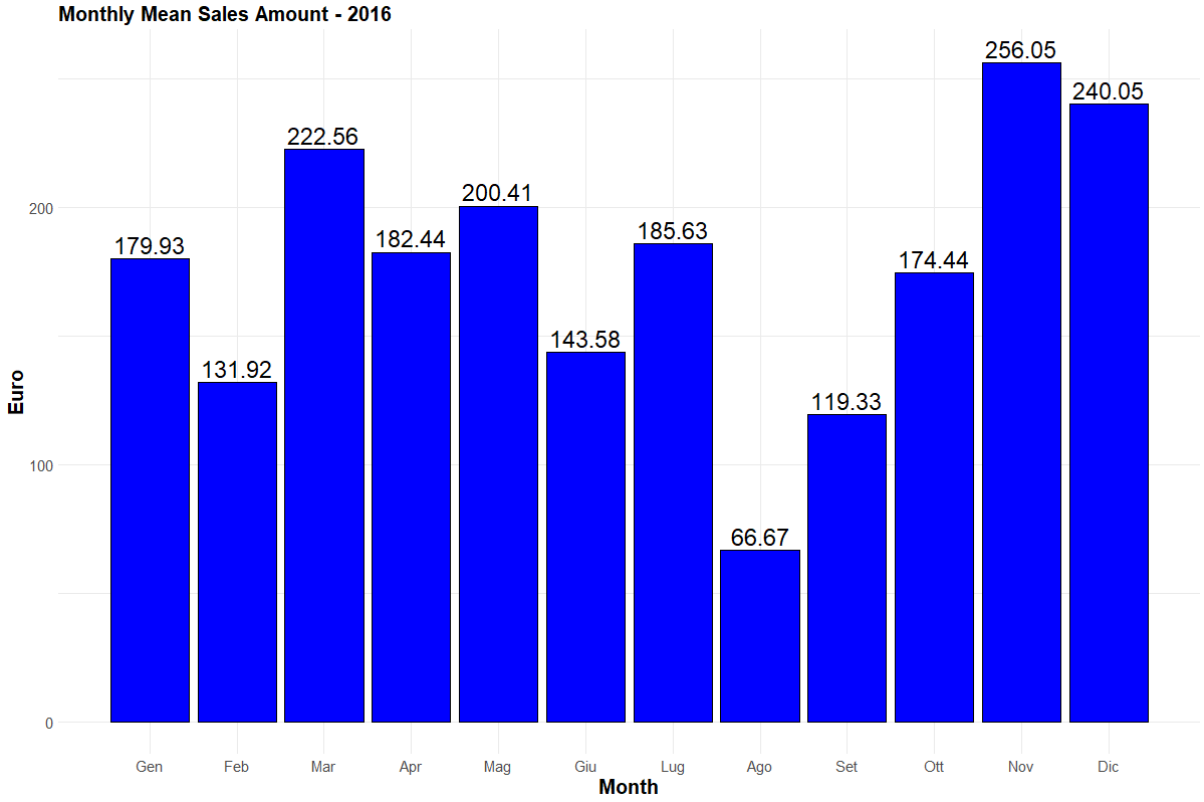
Output prodotti con R

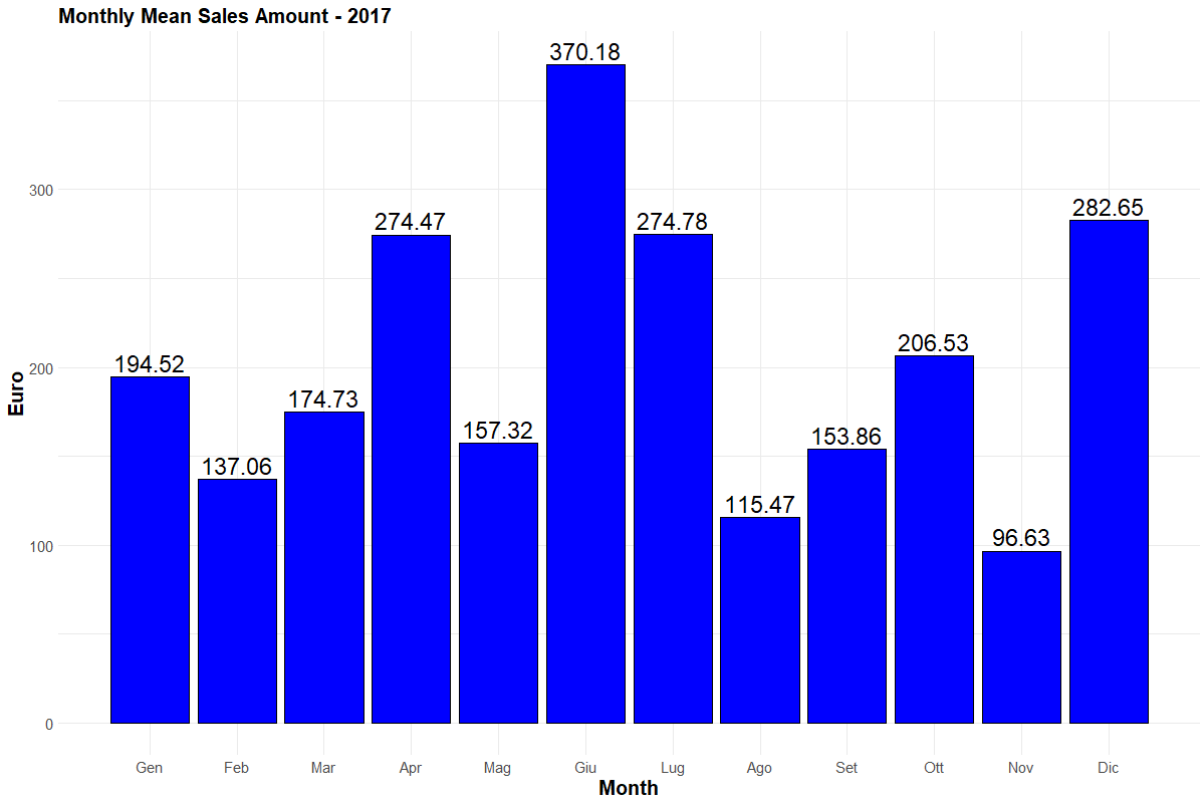
output/j1.csv	output/j2.csv
"month","value"	"month","value"
"01-2016",179.93	"01-2016",109182.809088059
"02-2016",131.92	"02-2016",37769.9670932277
"03-2016",222.56	"03-2016",290493.276026909
"04-2016",182.44	"04-2016",188377.840045667
"05-2016",200.41	"05-2016",193220.814067322
"06-2016",143.58	"06-2016",68699.7069424658
"07-2016",185.63	"07-2016",158191.320873529
"08-2016",66.67	"08-2016",5458.79683129252
"09-2016",119.33	"09-2016",34899.2390733611
"10-2016",174.44	"10-2016",150478.958366622
"11-2016",256.05	"11-2016",383866.658861879
"12-2016",240.05	"12-2016",251288.916418311
"01-2017",194.52	"01-2017",178609.971532785
"02-2017",137.06	"02-2017",42604.8812438347
"03-2017",174.73	"03-2017",87043.1328939358
"04-2017",274.47	"04-2017",456379.480631934
"05-2017",157.32	"05-2017",38106.4243999375
"06-2017",370.18	"06-2017",610883.168050331
"07-2017",274.78	"07-2017",252680.171231556
"08-2017",115.47	"08-2017",12353.50494475
"09-2017",153.86	"09-2017",42767.7282877909
"10-2017",206.53	"10-2017",674626.873414738
"11-2017",96.63	"11-2017",7192.30993017284
"12-2017",282.65	"12-2017",431920.895706841
"01-2018",144.47	"01-2018",37468.2903627066
"02-2018",152.43	"02-2018",49331.7906197239
"03-2018",190.62	"03-2018",138974.293707561
"04-2018",248.88	"04-2018",310487.290157599
"05-2018",158.39	"05-2018",67803.1509372761
"06-2018",176.2	"06-2018",203927.187488819

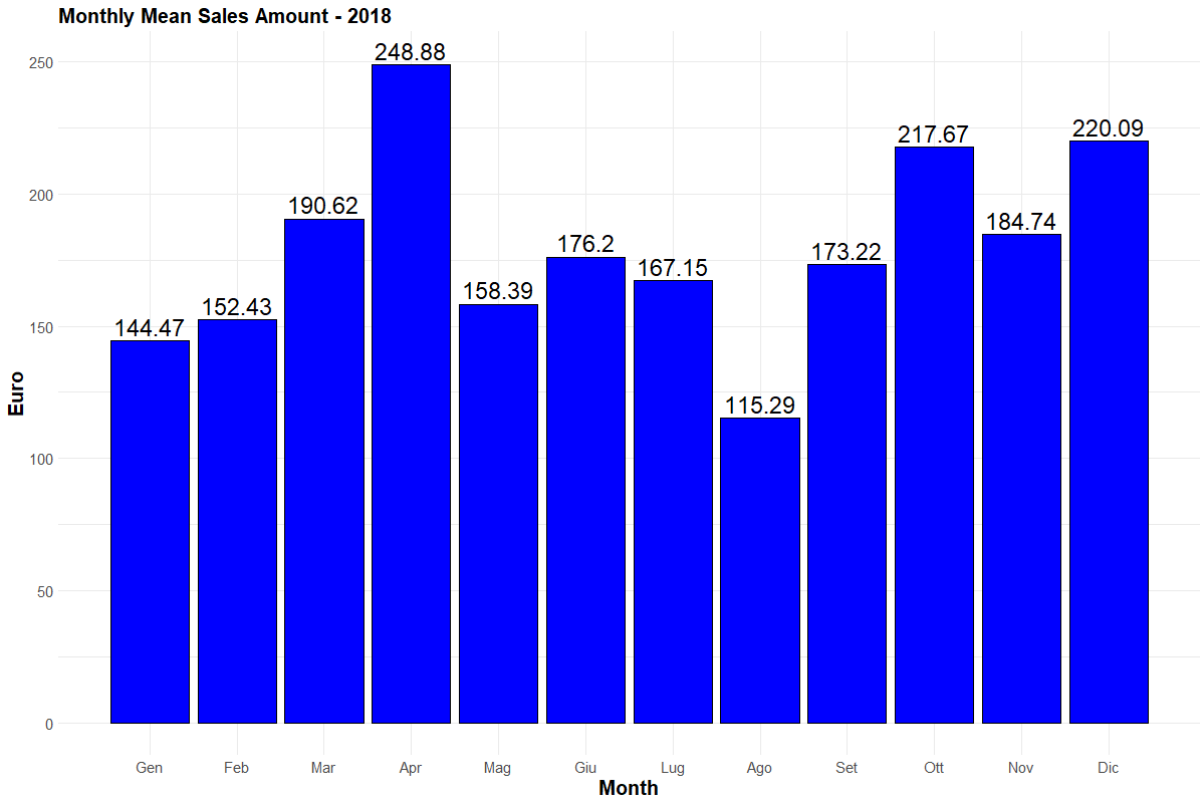
"07-2018",167.15	"07-2018",136784.957891623
"08-2018",115.29	"08-2018",7420.78990311419
"09-2018",173.22	"09-2018",226238.557854705
"10-2018",217.67	"10-2018",264467.546172245
"11-2018",184.74	"11-2018",76820.7610726331
"12-2018",220.09	"12-2018",207918.920680992
"01-2019",194.41	"01-2019",205325.0016862
"02-2019",193.96	"02-2019",56916.9088363636
"03-2019",128.86	"03-2019",86870.0589592825
"04-2019",243.09	"04-2019",204762.493094587
"05-2019",263.93	"05-2019",240459.236424985
"06-2019",233.3	"06-2019",100205.257356842
"07-2019",177.07	"07-2019",121202.557014124
"08-2019",366.83	"08-2019",286759.05375616
"09-2019",239.87	"09-2019",156084.550895544
"10-2019",223.59	"10-2019",192424.839445408
"11-2019",193.84	"11-2019",214675.515259967
"12-2019",193.75	"12-2019",174589.69305917
"01-2020",102.03	"01-2020",20222.4706478733
"02-2020",134.48	"02-2020",15642.8392541103
"03-2020",143.34	"03-2020",31183.0782782222
"05-2020",92.96	"05-2020",12501.6235957151
"06-2020",224.31	"06-2020",228708.358562585
"07-2020",210.94	"07-2020",86087.257747666
"08-2020",289.98	"08-2020",219139.98769566

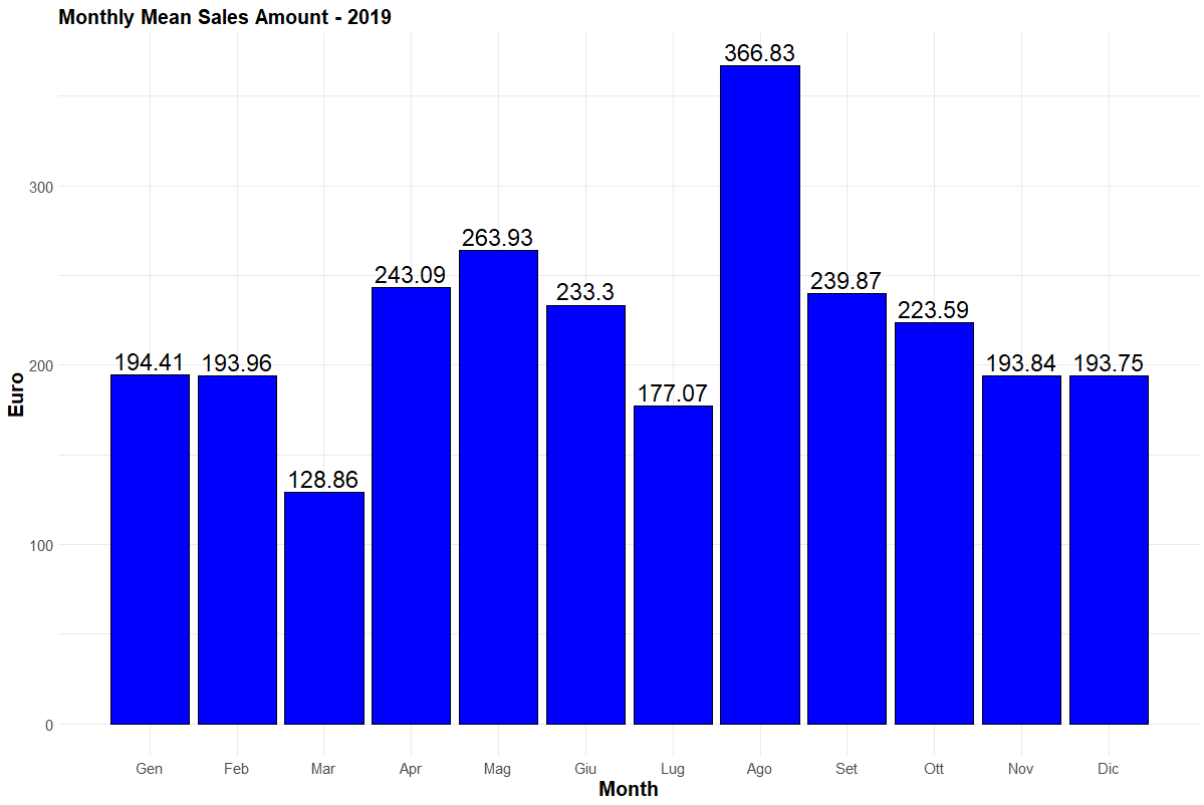
output/j3.csv	output/j4.csv
"month","value"	"month","value"
"12-2016",16083.28	"08-2016",1400.04
"06-2017",17028.09	"08-2017",2309.31
"04-2018",10701.99	"08-2018",1959.88
"10-2019",12520.92	"02-2019",6400.68
"06-2020",9420.84	"03-2020",2150.03

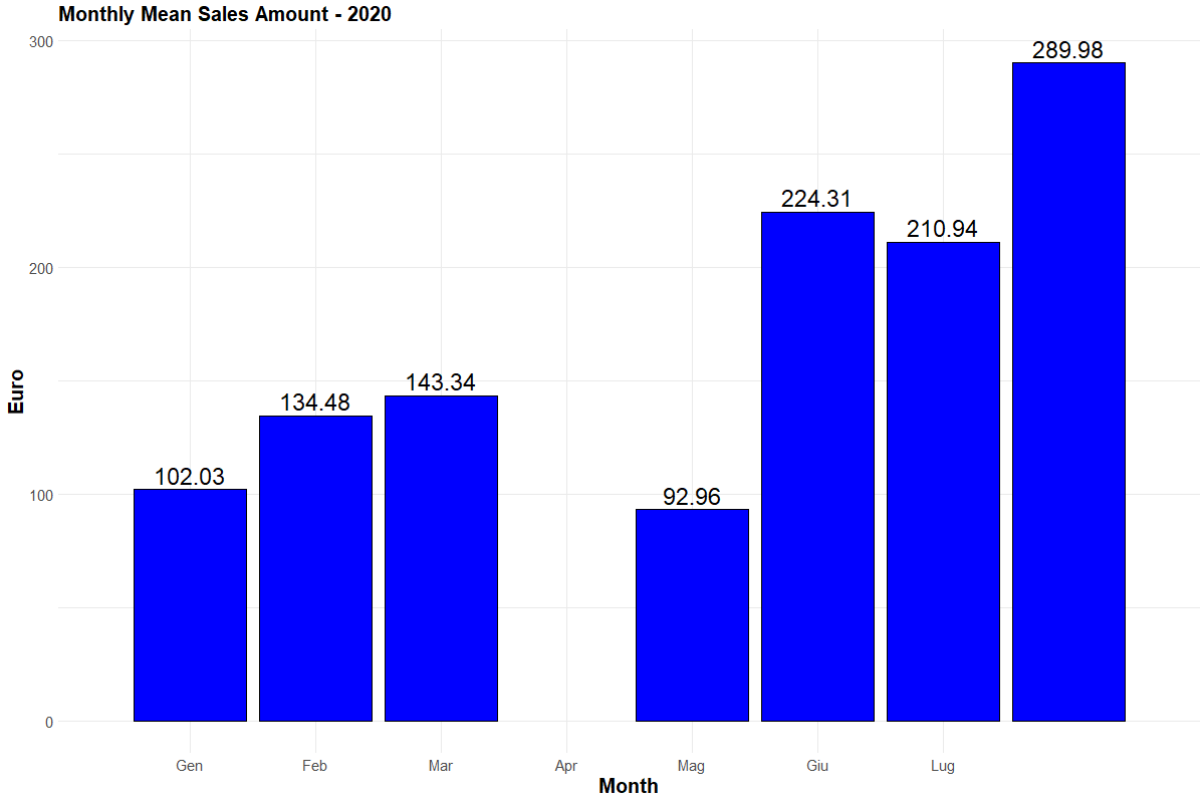
Grafici prodotti con R

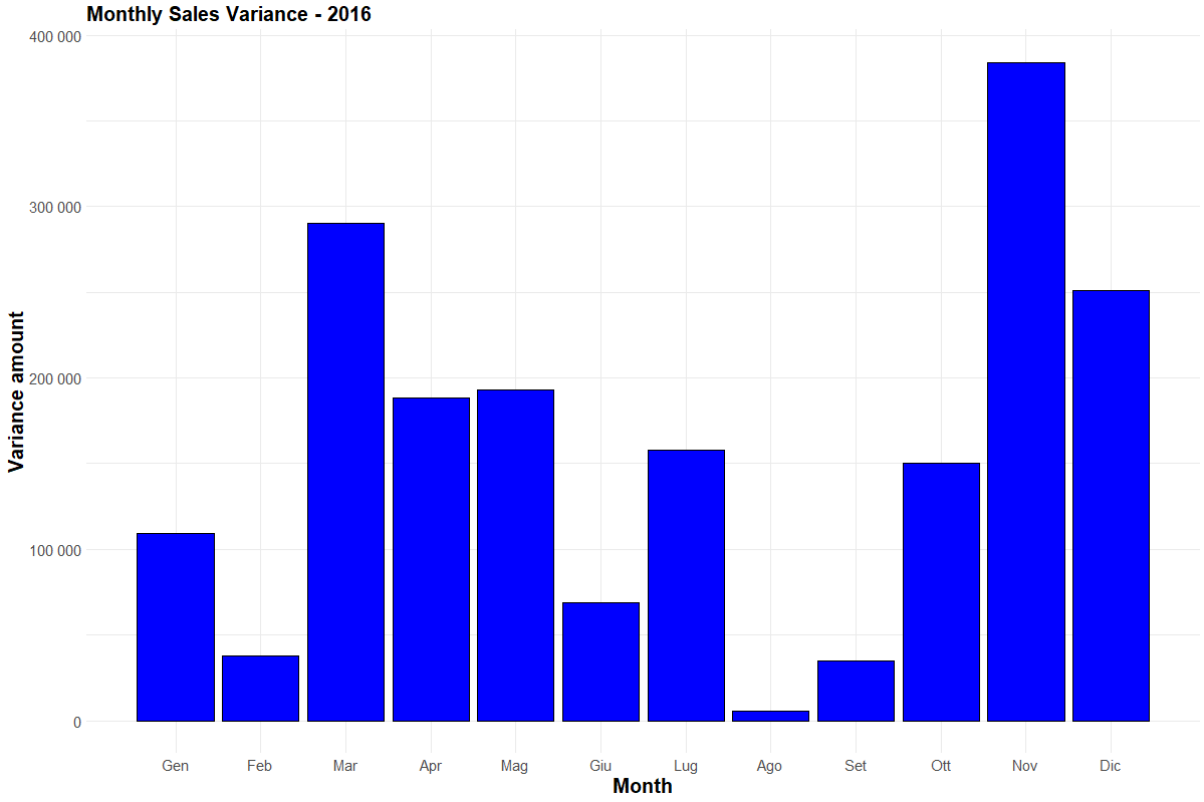


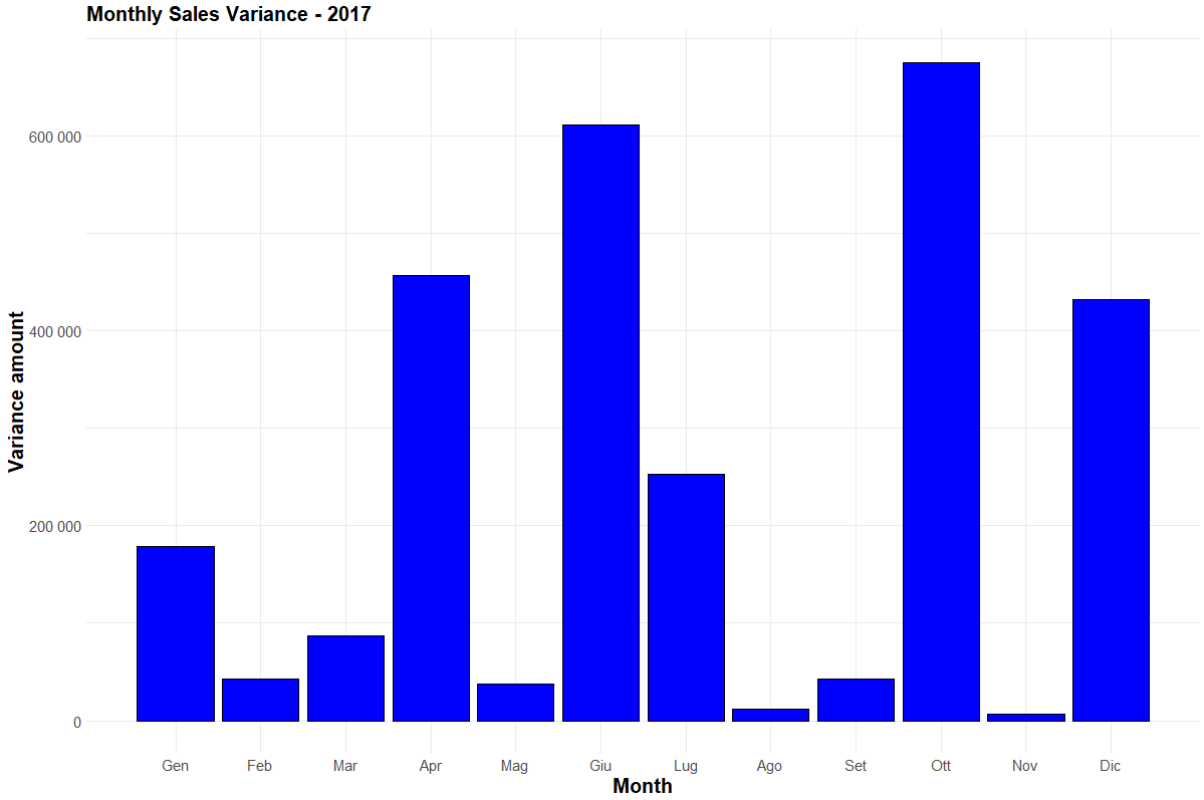


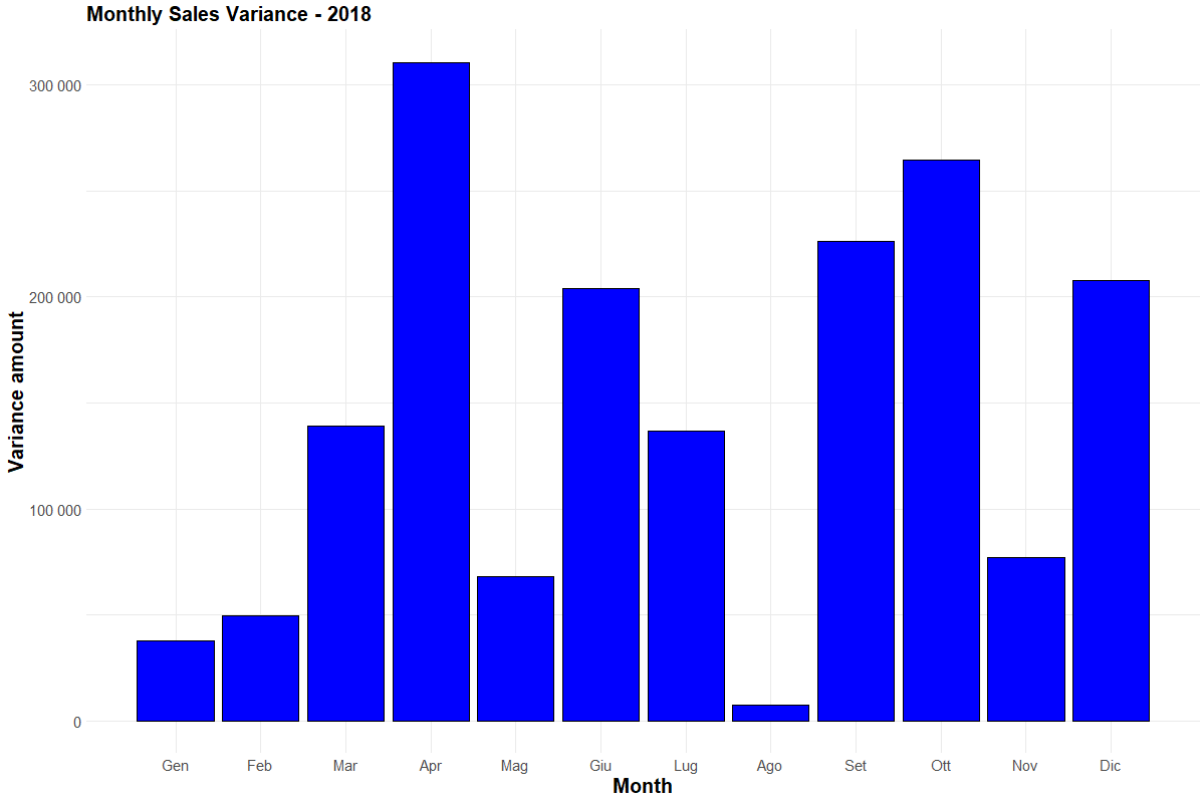

Monthly Mean Sales Amount - 2018
Euro
Month
0
50
100
150
200
250
144.47
152.43
190.62
248.88
158.39
176.2
167.15
115.29
173.22
217.67
184.74
220.09
Gen
Feb
Mar
Apr
Mag
Giu
Lug
Ago
Set
Ott
Nov
Dic

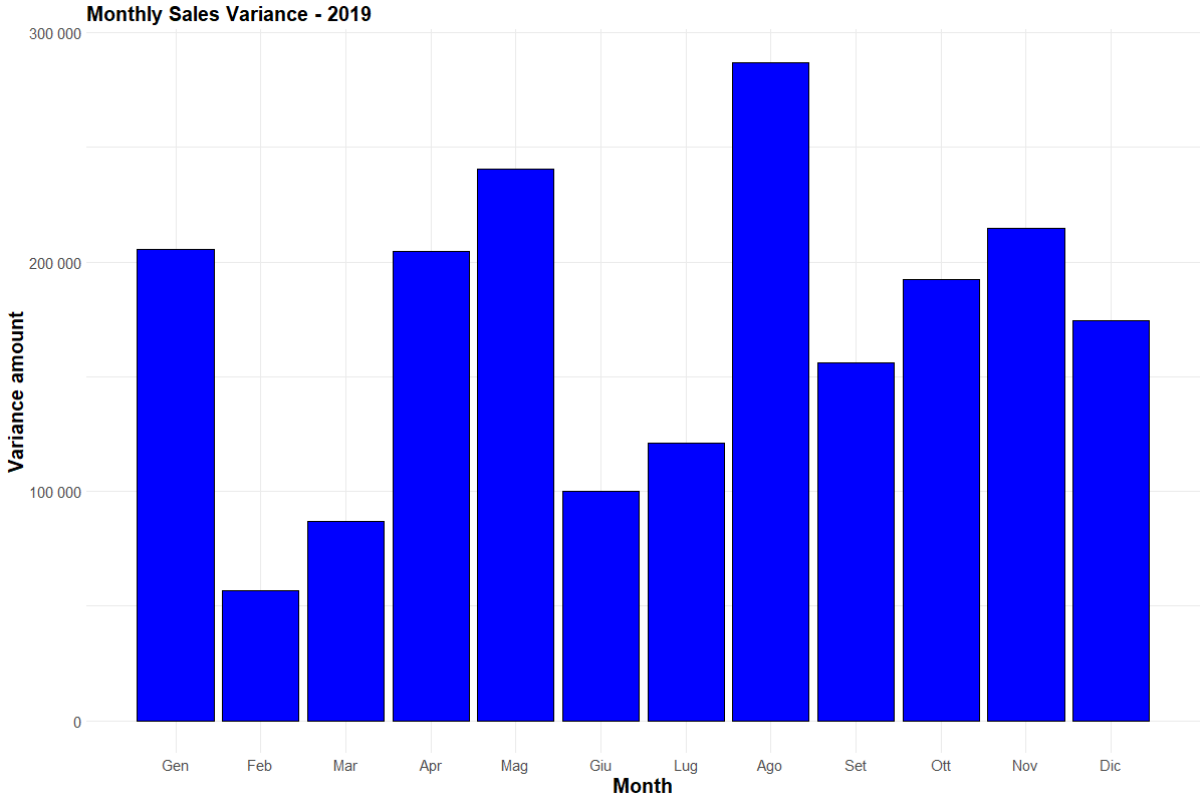

Monthly Mean Sales Amount - 2019
Euro
Month
0
100
200
300
194.41
193.96
128.86
243.09
263.93
233.3
177.07
366.83
239.87
223.59
193.84
193.75
Gen
Feb
Mar
Apr
Mag
Giu
Lug
Ago
Set
Ott
Nov
Dic

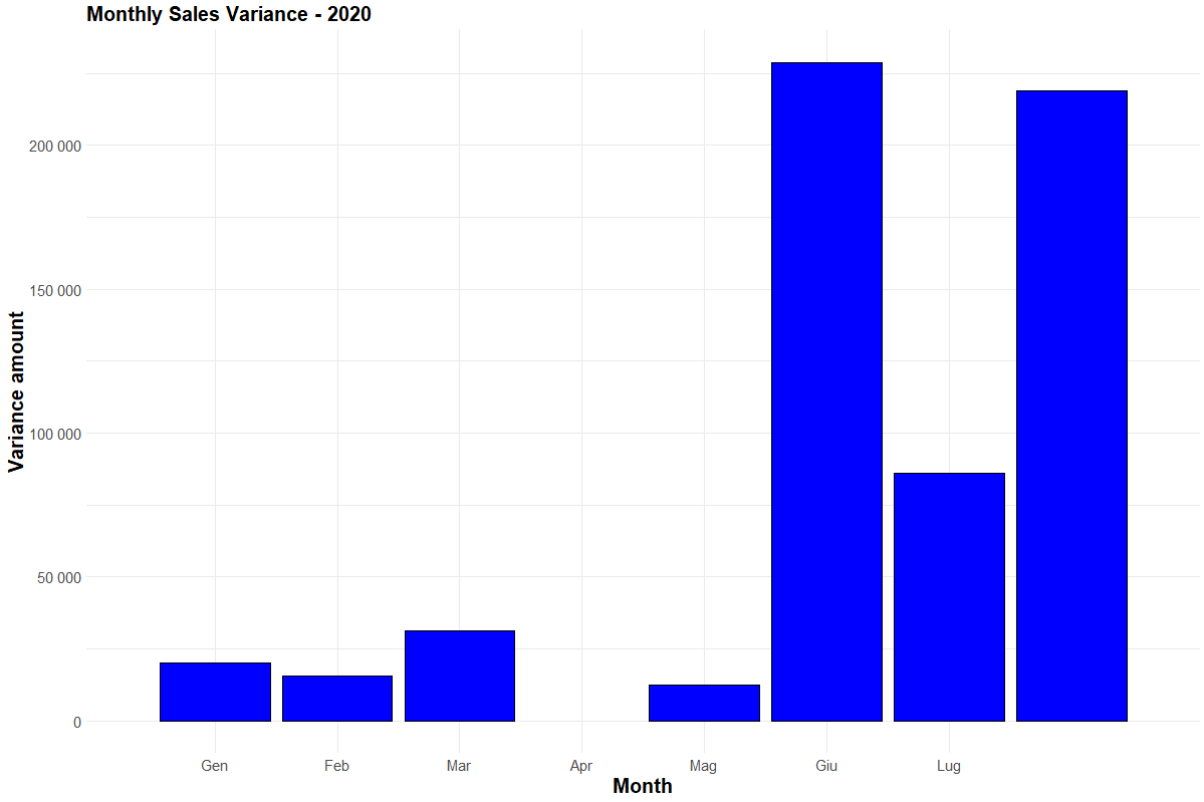

Monthly Mean Sales Amount - 2020
Euro
300
200
100
0
102.03
134.48
143.34
92.96
224.31
210.94
289.98
Gen
Feb
Mar
Apr
Mag
Giu
Lug
Month

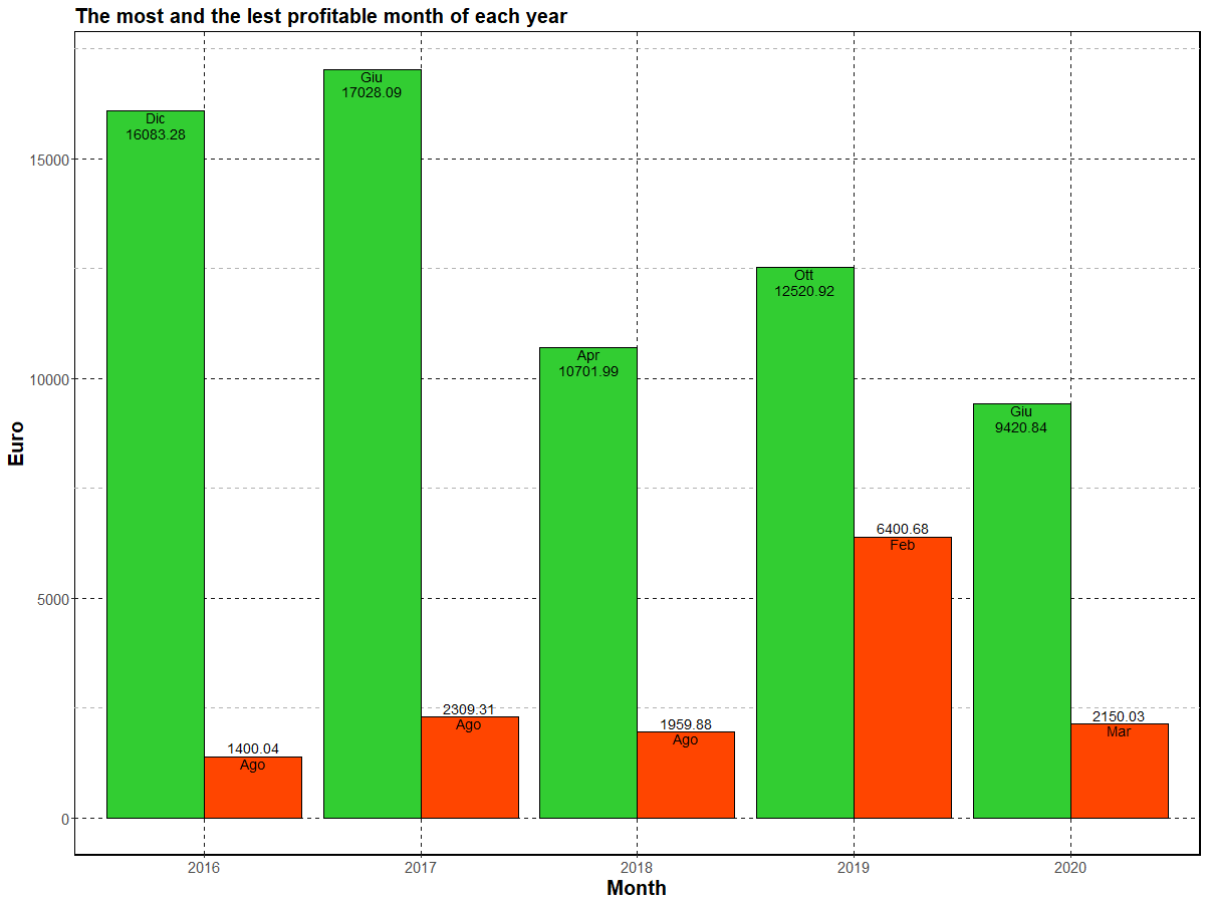

Monthly Sales Variance - 2016
Variance amount
400 000
300 000
200 000
100 000
0
Gen
Feb
Mar
Apr
Mag
Giu
Lug
Ago
Set
Ott
Nov
Dic
Month

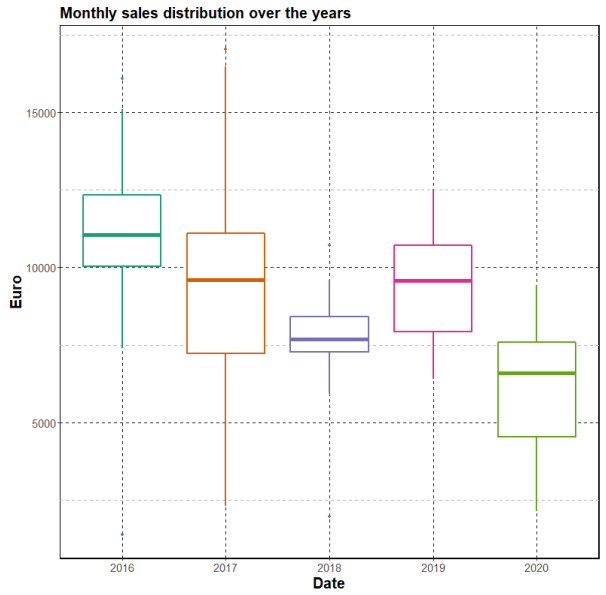

Monthly Sales Variance - 2017
Variance amount
600 000
400 000
200 000
0
Gen
Feb
Mar
Apr
Mag
Giu
Lug
Ago
Set
Ott
Nov
Dic
Month

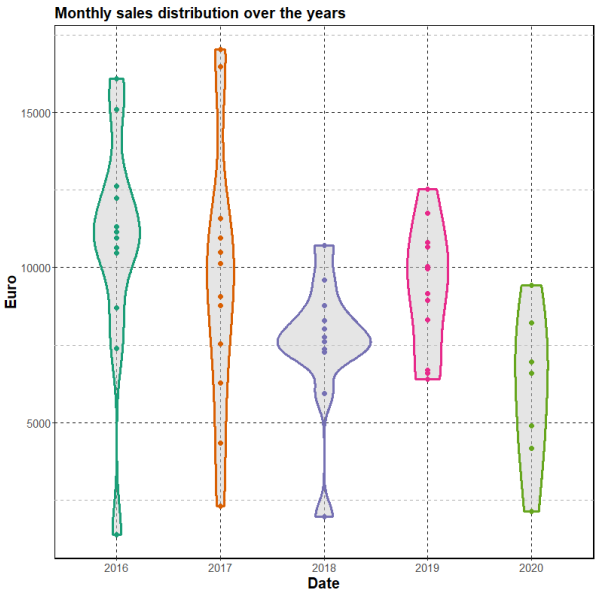

Monthly Sales Variance - 2018
Variance amount
300 000
200 000
100 000
0
Gen
Feb
Mar
Apr
Mag
Giu
Lug
Ago
Set
Ott
Nov
Dic
Month

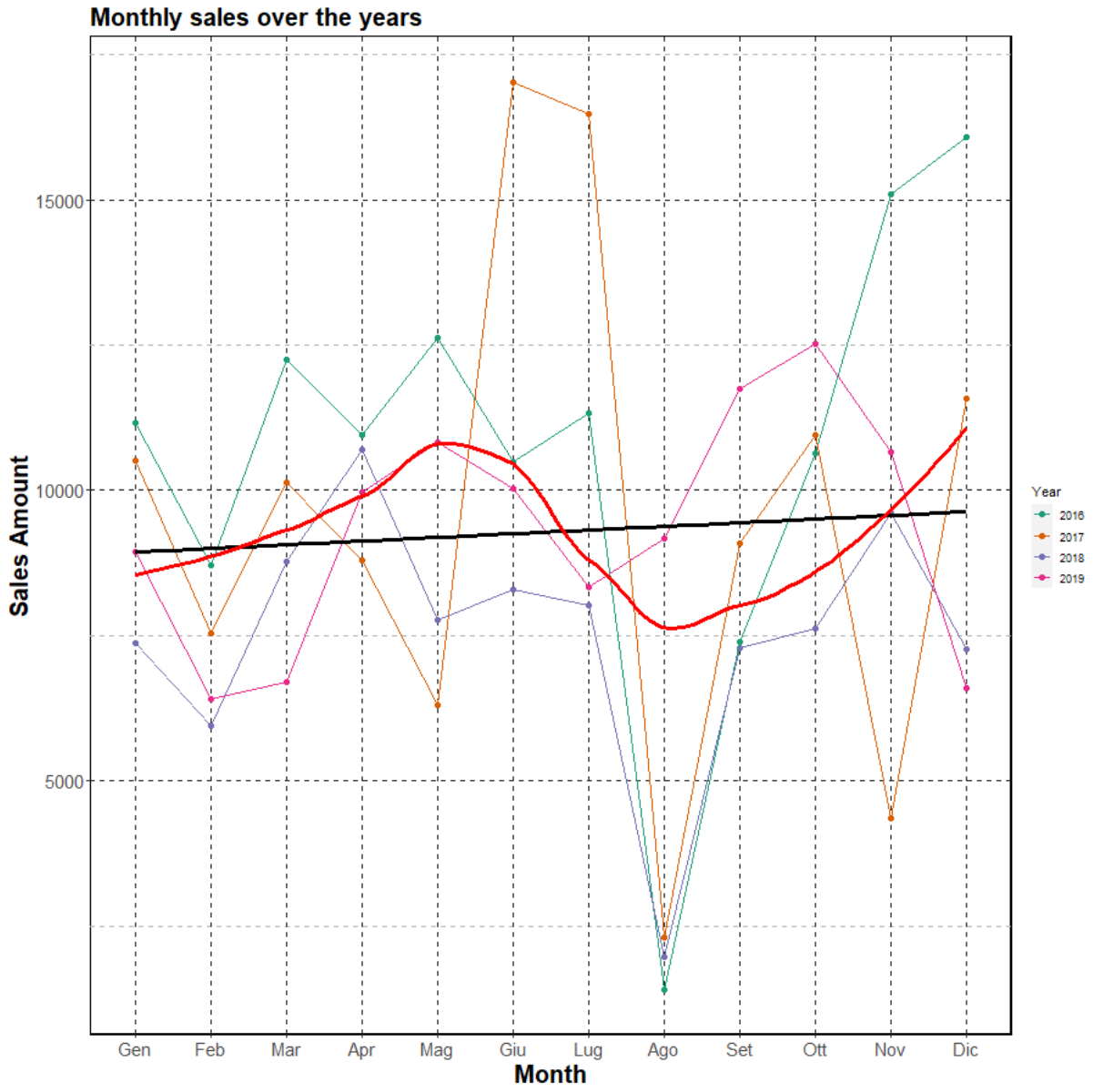

Monthly Sales Variance - 2019
300 000
200 000
100 000
0
Variance amount
Gen
Feb
Mar
Apr
Mag
Giu
Lug
Ago
Set
Ott
Nov
Dic
Month


Monthly Sales Variance - 2020
200 000
150 000
100 000
50 000
0
Variance amount
Gen
Feb
Mar
Apr
Mag
Giu
Lug
Month


The most and the lest profitable month of each year
Dic
16083.28
1400.04
Ago
Giu
17028.09
2309.31
Ago
Apr
10701.99
1959.88
Ago
Ott
12520.92
6400.68
Feb
Giu
9420.84
2150.03
Mar
15000
10000
5000
0
Euro
2016
2017
2018
2019
2020
Month


Monthly sales distribution over the years
15000
10000
5000
Euro
2016
2017
2018
2019
2020
Date


Monthly sales distribution over the years
15000
10000
5000
Euro
2016
2017
2018
2019
2020
Date



Codice jobs Hadoop con R

j1/mapper.r

```
1.  # declaring utility functions
2.  trim_white_space <- function(line) gsub("(^ +)|( +$)", "", line)
3.  split_into_fields <- function(line) unlist(strsplit(line, ","))
4.  filter_type <- c("FATTURA", "RICEVUTA")
5.
6.  # reading what is in the console
7.  con <- file("stdin", open = "r")
8.  # iterating through the lines
9.  while (length(line <- readLines(con, n = 1, warn = FALSE)) > 0) {
10.   line <- trim_white_space(line)
11.   # splitting the line into fields
12.   fields <- split_into_fields(line)
13.
14.   # checking if the type is FATTURA or RICEVUTA
```

```
15.   # and output string
16.   if (fields[1] %in% filter_type)
17.     cat(substr(fields[2], 1, 6), "\t", fields[3], "\n", sep = "")
18. }
19. close(con)
```

j1/reducer.r

```
1.  # declaring utility functions
2.  trim_white_space <- function(line) gsub("(^ +)|( +$)", "", line)
3.  split_line <- function(line) {
4.    val <- unlist(strsplit(line, "\t"))
5.    list(month = val[1], euro = as.numeric(val[2]))
6.  }
7.
8.  # creating a new environment
9.  env <- new.env(hash = TRUE)
10. # reading what is in the console
11. con <- file("stdin", open = "r")
12. # iterating through the lines
13. while (length(line <- readLines(con, n = 1, warn = FALSE)) > 0) {
14.   line <- trim_white_space(line)
15.   # splitting the line into fields to get a list describing the sale
16.   sale <- split_line(line)
17.   new_value <- list(tot  = 0, count = 0)
18.   # checking if the month is already in the environment
19.   if (exists(sale$month, envir = env, inherits = FALSE)) {
20.       # get the current value
21.       old_value <- get(sale$month, envir = env)
22.       # update the count and the total
23.       new_value$count <- old_value$count + 1
24.       new_value$tot <- old_value$tot + sale$euro
25.   }
26.   else {
27.     # setup the count and the total
28.     new_value$count <- 1
29.     new_value$tot <- sale$euro
30.   }
31.   # assign the new value to the month into the environment
32.   assign(sale$month, new_value, envir = env)
33. }
34. close(con)
35.
36. # iterating through the months in the environment
37. # and writing output
38. for (month in ls(env, all = TRUE)) {
39.   value <- get(month, envir = env)
40.   cat(month, "\t", value$tot / value$count, "\n", sep = "")
41. }
```

j1/cmd.sh

```
1.  work_folder=/home/hdoop/R/src/prj
2.  # deleting the folder to ensure it doesn't exist
3.  hdfs dfs -rm -r /prj/j1
```

```
# creating the job folder
hdfs dfs -mkdir /prj/j1
# copying the data to the hdfs
hdfs dfs -copyFromLocal $work_folder/Ordini.csv /prj/j1

# changing the permissions of the R scripts
cd $work_folder
chmod 777 $work_folder/j1/mapper.r $work_folder/j1/reducer.r

# running the job via hadoop streaming
hadoop jar /home/hdoop/Documents/hadoop-streaming-3.2.1.jar \
-file /home/hdoop/R/src/prj/j1/mapper.r  -mapper 'Rscript
   /home/hdoop/R/src/prj/j1/mapper.r' \
-file /home/hdoop/R/src/prj/j1/reducer.r -reducer 'Rscript
   /home/hdoop/R/src/prj/j1/reducer.r' \
-input /prj/j1/Ordini.csv -output /prj/j1/output

# showing the output
hdfs dfs -cat /prj/j1/output/part-00000
```

j2/mapper.r

```
# Per il codice di j2/mapper.r fare riferimento a j1/mapper.r.
```

j2/reducer.r

```
library(data.table)

# declaring utility functions
trim_white_space <- function(line) gsub("(^ +)|( +$)", "", line)
split_line <- function(line) {
  val <- unlist(strsplit(line, "\t"))
  list(month = val[1], euro = as.numeric(val[2]))
}
calculate_sqm <- function(value, month) {
  return((value - mean_by_month[mean_by_month$V1 == month]$V2)^2)
}

# reading the mean values stored in hdfs calculated by JOB 1
mean_by_month <- fread("hadoop fs -text /prj/j1/output/part-00000")
# creating a new environment
env <- new.env(hash = TRUE)
# reading what is in the console
con <- file("stdin", open = "r")
# iterating through the lines
while (length(line <- readLines(con, n = 1, warn = FALSE)) > 0) {
  line <- trim_white_space(line)
  # splitting the line into fields to get a list describing the sale
  sale <- split_line(line)
  new_value <- list(tot  = 0, count = 0)
  # checking if the month is already in the environment
  if (exists(sale$month, envir = env, inherits = FALSE)) {
      # get the current value
      old_value <- get(sale$month, envir = env)
```

```
29.       # update the count and the summation of the squared differences
30.       new_value$count <- old_value$count + 1
31.       new_value$tot <- old_value$tot + calculate_sqm(sale$euro, sale$month)
32.   } else {
33.     # setup the count and the total
34.     new_value$count <- 1
35.     new_value$tot <- calculate_sqm(sale$euro, sale$month)
36.   }
37.   # assign the new value to the month into the environment
38.   assign(sale$month, new_value, envir = env)
39. }
40. close(con)
41.
42. # iterating through the months in the environment
43. # and writing output
44. for (month in ls(env, all = TRUE)) {
45.   value <- get(month, envir = env)
46.   cat(month, "\t", toString(value$tot / value$count), "\n", sep = "")
47. }
```

j2/cmd.sh

```
# Per il codice di j2/cmd.sh fare riferimento a j1/cmd.sh sostituendo tutte le
occorrenze di “j1” con “j2”.
```

j3/mapper.r

```
# Per il codice di j3/mapper.r fare riferimento a j1/mapper.r.
```

j3/reducer.r

```
1.  # declaring utility functions
2.  trim_white_space <- function(line) gsub("(^ +)|( +$)", "", line)
3.  split_line <- function(line) {
4.    val <- unlist(strsplit(line, "\t"))
5.    list(month = val[1], euro = as.numeric(val[2]))
6.  }
7.
8.  # creating a new environment
9.  env <- new.env(hash = TRUE)
10. # reading what is in the console
11. con <- file("stdin", open = "r")
12. # iterating through the lines
13. while (length(line <- readLines(con, n = 1, warn = FALSE)) > 0) {
14.   line <- trim_white_space(line)
15.   # splitting the line into fields to get a list describing the sale
16.   sale <- split_line(line)
17.   # assign the new value to the month into the environment
18.   # the assigned values is:
19.   # the sum of old and new value, if the month is already in the environment
20.   # the new value, otherwise
21.   assign(sale$month,
```

```
22.     ifelse(exists(sale$month, envir = env, inherits = FALSE),
23.       get(sale$month, envir = env) + sale$euro,
24.       sale$euro),
25.     envir = env)
26. }
27.
28. # getting all the months in the environment
29. months <- ls(env, all = TRUE)
30. # creating a dataframe to store the months and sum of of their sale amounts
31. values <- data.frame()
32. for (month in months) {
33.   values <- rbind(values, c(month, get(month, envir = env)))
34. }
35. colnames(values) <- c("month", "euro")
36.
37. # iterating through the years
38. for (year in unique(substring(months, 1, 4))) {
39.   # filtering all the months in the year
40.   values_in_year <- values[which(startsWith(values$month, year)), ]
41.   # getting the month with the highest sale amount
42.   max <- values_in_year[which.max(values_in_year$euro), ]
43.   # writing output
44.   cat(max$month, "\t", max$euro, "\n", sep = "")
45. }
```

j3/cmd.sh

```
# Per il codice di j3/cmd.sh fare riferimento a j1/cmd.sh sostituendo tutte le
occorrenze di "j1" con "j3".
```

j4/mapper.r

```
# Per il codice di j4/mapper.r fare riferimento a j1/mapper.r.
```

j4/reducer.r

```
# Per il codice di j4/reducer.r fare riferimento a j3/reducer.r da riga 1 a riga
36.
# Il codice di j4/reducer.r continua come segue.
37. # iterating through the years
38. for (year in unique(substring(months, 1, 4))) {
39.   # filtering all the months in the year
40.   values_in_year <- values[which(startsWith(values$month, year)), ]
41.   # getting the month with the lowest sale amount
42.   min <- values_in_year[which.min(values_in_year$euro), ]
43.   # writing output
44.   cat(min$month, "\t", min$euro, "\n", sep = "")
45. }
```

j4/cmd.sh

```
# Per il codice di j4/cmd.sh fare riferimento a j1/cmd.sh sostituendo tutte le occorrenze di "j1" con "j4".
```

Output prodotti con Hadoop e R

/prj/j1/output/part-0000

```
201601  179.9308
201602  131.9229
201603  222.558
201604  182.441
201605  200.4127
201606  143.58
201607  185.6295
201608  66.66857
201609  119.3348
201610  174.442
201611  256.0469
201612  240.049
201701  194.5207
201702  137.0633
201703  174.7252
201704  274.4684
201705  157.3173
201706  370.1759
201707  274.7847
201708  115.4655
201709  153.8563
201710  206.5275
201711  96.63378
201712  282.6517
201801  144.4731
201802  152.4346
201803  190.6152
201804  248.8835
201805  158.3888
201806  176.2049
201807  167.1452
201808  115.2871
201809  173.2198
```

/prj/j2/output/part-0000

```
201601  109182.809088059
201602  37769.9670932282
201603  290493.276026909
201604  188377.840045667
201605  193220.814067322
201606  68699.7069424657
201607  158191.320873529
201608  5458.79683129252
201609  34899.2390733626
201610  150478.958366623
201611  383866.658861881
201612  251288.916418313
201701  178609.971532786
201702  42604.8812438355
201703  87043.1328939365
201704  456379.480631935
201705  38106.42439994
201706  610883.168050332
201707  252680.171231557
201708  12353.50494475
201709  42767.7282877917
201710  674626.873414741
201711  7192.30993017284
201712  431920.895706841
201801  37468.290362708
201802  49331.7906197241
201803  138974.293707562
201804  310487.290157599
201805  67803.1509372767
201806  203927.187488819
201807  136784.957891623
201808  7420.78990311588
201809  226238.557854707
```

201810 217.6714	201810 264467.546172246
201811 184.7365	201811 76820.7610726346
201812 220.0909	201812 207918.920680992
201901 194.4091	201901 205325.001686201
201902 193.96	201902 56916.9088363636
201903 128.8644	201903 86870.0589592831
201904 243.0893	201904 204762.493094588
201905 263.9251	201905 240459.236424986
201906 233.3033	201906 100205.257356843
201907 177.0717	201907 121202.557014124
201908 366.8272	201908 286759.05375616
201909 239.8706	201909 156084.550895544
201910 223.5879	201910 192424.83944541
201911 193.8362	201911 214675.515259967
201912 193.7476	201912 174589.693059172
202001 102.0298	202001 20222.4706478733
202002 134.4768	202002 15642.839254111
202003 143.3353	202003 31183.0782782233
202005 92.95606	202005 12501.6235957151
202006 224.3057	202006 228708.358562585
202007 210.9356	202007 86087.2577476677
202008 289.9796	202008 219139.98769566

/prj/j3/output/part-0000	/prj/j4/output/part-0000
201612 16083.28	201608 1400.04
201706 17028.09	201708 2309.31
201804 10701.99	201808 1959.88
201910 12520.92	201902 6400.68
202006 9420.84	202003 2150.03