hotel-bookings

June 11, 2020

1 Hotel Bookings

Programmazione di Applicazioni Data Intensive
Laurea in Ingegneria e Scienze Informatiche
DISI - Università di Bologna, Cesena

Studente: Alessandro Lombardini
alessandr.lombardin3@unibo.it

[335]:
```
# Setup librerie
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

1.1 Caso di studio

- Data una prenotazione di un cliente presso un hotel, si vuole valutare se questa verrà cancellata
- Da ciascuna prenotazione possono essere estratte delle caratteristiche
  - numero di adulti, giorno di arrivo, numero di posti auto richiesti, ...
- Vogliamo addestrare un modello a classificare ciascuna prenotazione sulla base di queste caratteristiche
- Utilizziamo Hotel booking demand, in cui ogni osservazione contiene le caratteristiche estratte da una prenotazione
- Con `read_csv` possiamo importare il dataset direttamente in un frame pandas dato il suo URL

[336]:
```
HBD_URL = "https://bitbucket.org/alessandrolombardini/hotel-bookings/raw/
↪d5130f7e47e3e943886c1ab5266197202e6fe9af/hotel_bookings.csv"
hbd_complete = pd.read_csv(HBD_URL)
```

[337]:
```
hbd_complete.head(3)
```

[337]:

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	\
0	Resort Hotel	0	342	2015	July	
1	Resort Hotel	0	737	2015	July	

```
2  Resort Hotel            0          7               2015               July

   arrival_date_week_number  arrival_date_day_of_month  \
0                        27                          1
1                        27                          1
2                        27                          1

   stays_in_weekend_nights  stays_in_week_nights  adults  children  babies  \
0                        0                     0       2       0.0       0
1                        0                     0       2       0.0       0
2                        0                     1       1       0.0       0

  meal country market_segment distribution_channel  is_repeated_guest  \
0   BB     PRT         Direct               Direct                  0
1   BB     PRT         Direct               Direct                  0
2   BB     GBR         Direct               Direct                  0

   previous_cancellations  previous_bookings_not_canceled reserved_room_type  \
0                       0                               0                  C
1                       0                               0                  C
2                       0                               0                  A

  assigned_room_type  booking_changes deposit_type  agent  company  \
0                  C               3   No Deposit    NaN      NaN
1                  C               4   No Deposit    NaN      NaN
2                  C               0   No Deposit    NaN      NaN

   days_in_waiting_list customer_type   adr  required_car_parking_spaces  \
0                     0     Transient   0.0                            0
1                     0     Transient   0.0                            0
2                     0     Transient  75.0                            0

   total_of_special_requests reservation_status reservation_status_date
0                          0          Check-Out              2015-07-01
1                          0          Check-Out              2015-07-01
2                          0          Check-Out              2015-07-02
```

- `Hotel booking demand` prevede al suo interno due dataset, uno per *Resort Hotel* e uno per *City Hotel*
  - Entrambi i dataset condividono la stessa struttura
  - Entrambi i dataset comprendono prenotazioni effettuate dal 1 Luglio 2015 al 31 Agosto 2017
- Poichè questi sono dati reali, tutti i dati personali dei clienti sono stati eliminati oppore sostiuiti con identificativi anonimi
- Il dataset presenta le seguenti dimensioni…

[338]:
```
print(hbd_complete.shape[0], "istanze")
print(hbd_complete.shape[1], "variabili\n")
print(hbd_complete[hbd_complete["hotel"]=="City Hotel"].shape[0], "istanze di␣
↪City Hotel")
print(hbd_complete[hbd_complete["hotel"]=="Resort Hotel"].shape[0], "istanze di␣
↪Resort Hotel")
```

```
119390 istanze
32 variabili

79330 istanze di City Hotel
40060 istanze di Resort Hotel
```

- Il nostro obiettivo è realizzare un modello di classificazione per la struttura *City Hotel*
- Si tratta di classificazione binaria, ovvero con due possibili classi
  - La colonna `is_canceled` indica la classificazione delle prenotazioni
    - 0 = non cancellata
    - 1 = cancellata
  - Le altre 31 colonne corrispondono alle altre variabili estratte dalla prenotazione

1.1.1 Lista delle variabili

- `hotel`: hotel prenotato dal cliente (H1 = Resort Hotel o H2 = City Hotel)
- `lead_time`: numero di giorni che intercorrono dal giorno di prenotazione al giorno di arrivo in hotel del cliente
- `arrival_date_year`: anno di arrivo del cliente in hotel
- `arrival_date_month`: mese di arrivo del cliente in hotel
- `arrival_date_week_number`: numero della settimana dell'anno di arrivo del cliente in hotel
- `arrival_date_day_of_month`: numero del giorno del mese di arrivo del cliente in hotel
- `stays_in_weekend_nights`: numero di notti di finesettimana (Sabato e Domenica) prenotate del cliente
- `stays_in_week_nights`: numero di notti non di finesettimana (da Lunedi a Venerdi) prenotate del cliente
- `adults`: numero di adulti
- `children`: numero di bambini
- `babies`: numero di neonati
- `meal`: pacchetto pasti richiesto dal cliente
  - Undefined/SC – nessuno pacchetto
  - BB – Bed & Breakfast
  - HB – Mezza pensione (colazione ed un altro pasto – solitamente cena)

  - FB – Pensione completa (colazione, pranzo e cena)
- `country`: stato di provenienza
- `market_segment`: segmento di mercato associato alla prenotazione (utile per raggruppare le prenotazioni in gruppi, al fine di adottare stategie di marketing adeguate)
  - Online TA: Online Travel Agent
  - Offline TO: Offline Tour Operator
  - …
- `distribution_channel`: canale per tramite del quale il cliente ha effettuato la prenotazione
  - TA/TO: il cliente si è appoggiato ad un agente di viaggio
  - Direct: la prenotazione è stata fatta dal cliente direttamente
  - …
- `is_repeated_guest`: indica se la prenotazione è fatta da un cliente che aveva già prenotato in passato
  - 1: Si, aveva già prenotato
  - 0: No, non aveva mai prenotato
- `previous_cancellations`: numero di prenotazioni cancellate in passato dal cliente
- `previous_bookings_not_canceled`: numero di prenotazioni effettuate in passato dal cliente e non cancellate
- `reserved_room_type`: codice del tipo di stanza richiesta dal cliente
- `assigned_room_type`: codice del tipo di stanza assegnata alla prenotazione. A volte vengono assegnate stanze diverse da quelle riservate per motivi legati all'Hotel (es. overbooking) o per richiesta del cliente.
- `booking_changes`: numero di cambiamenti apportati alla prenotazione fino al momento del Check-In o della cancellazione
- `deposit_type`: indica se il cliente ha effettuato un deposito per garantirsi la prenotazione
  - No Deposit: nessun deposito è stato fatto
  - Non Refund: è stato pagato l'intero importo del soggiorno
  - Refundable: è stata pagata solo una parte dell'importo dell'intero soggiorno
- `agent`: ID dell'agente di viaggio che ha effettuato la prenotazione
- `company`: ID della compagnia che ha effettuato la prenotazione o che ha pagato la prenotazione.
- `days_in_waiting_list`: numero di giorni in cui la prenotazione è rimasta in lista di attesa prima di essere confermata al cliente
- `customer_type`: tipologia di prenotazione
  - Contract: la prenotazione è associata ad un contratto

  - Group: la prenotazione è associata ad un gruppo
  - Transient: la prenotazione non è parte ne di un gruppo ne di un contratto, e non è associata ad altre prenotazione Transient
  - Transient-party: la prenotazione è Transient ed è associata ad altre prenotazioni Transient (Transient è un termine utilizzato per indicare quelle prenotazioni effettuate da soggetti prevalentemente in movimento che effettuano brevi soggiorni in hotel, spesso last minute)
- `adr`: Avarage Daily Rate, definito come il costo del soggiorno diviso il numero di notti
- `required_car_parking_spaces`: numero di spazi macchina richiesti dal cliente
- `total_of_special_requests`: numero di richieste speciali effettuate dal cliente
- `reservation_status`: ultimo stato registrato della prenotazione
  - Canceled: la prenotazione è stata cancellata dal cliente
  - Check-Out: il client ha effettuato il Check-In e la sua permanenza è terminata
  - No-Show: il cliente non ha effettuato il Check-In e ha informato l'hotel del motivo
- `reservation_status_date`: data dell'ultima modifica alla variabile `reservation_status`
- `is_canceled`: indica se la prenotazione è stata cancellata o no
  - 0: Non cancellata
  - 1: Cancellata
- La variabile **is_canceled** indica la classificazione della prenotazione, vogliamo stabilire il valore di questa variabile in funzione delle altre
- Aumentiamo il limite di colonne che pandas di default ci consente di visualizzare

[339]:
```
pd.options.display.max_columns = 32
```

[340]:
```
hbd_complete.head(3)
```

[340]:

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day_of_month	stays_in_weekend_nights	stays_in_week_nights	adults	children	babies
0	Resort Hotel	0	342	2015	July	27	1	0	0	2	0.0	0
1	Resort Hotel	0	737	2015	July	27	1	0	0	2	0.0	0
2	Resort Hotel	0	7	2015	July	27	1	0	1	1	0.0	0

```
  meal country market_segment distribution_channel  is_repeated_guest  \
0   BB     PRT         Direct               Direct                  0
1   BB     PRT         Direct               Direct                  0
2   BB     GBR         Direct               Direct                  0

   previous_cancellations  previous_bookings_not_canceled reserved_room_type  \
0                       0                               0                 C
1                       0                               0                 C
2                       0                               0                 A

  assigned_room_type  booking_changes deposit_type  agent  company  \
0                  C               3   No Deposit    NaN      NaN
1                  C               4   No Deposit    NaN      NaN
2                  C               0   No Deposit    NaN      NaN

   days_in_waiting_list customer_type   adr  required_car_parking_spaces  \
0                     0     Transient   0.0                            0
1                     0     Transient   0.0                            0
2                     0     Transient  75.0                            0

   total_of_special_requests reservation_status reservation_status_date
0                          0          Check-Out              2015-07-01
1                          0          Check-Out              2015-07-01
2                          0          Check-Out              2015-07-02
```

1.2 Preparazione dei dati

- Riportiamo lo spazio occupato in memoria dal dataset

[341]:
```
hbd_complete.info(verbose=False, memory_usage="deep");
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 119390 entries, 0 to 119389
Columns: 32 entries, hotel to reservation_status_date
dtypes: float64(4), int64(16), object(12)
memory usage: 105.7 MB
```

- Osserviamo che sono presenti molte variabili di tipo *object*
  - Il dataset così caricato occupa molto spazio, è quindi opportuno specificare che parte delle variabili *object* devono essere gestite come categoriche
    - Tutte eccetto `reservation_status_date` in quanto presenta come valori delle date

[342]:
```
object_variable = hbd_complete.dtypes[hbd_complete.dtypes == np.object].
↪drop(["reservation_status_date"]).index
hbd_complete[object_variable] = hbd_complete[object_variable].astype("category")
```

- Verifichiamo la dimensione in memoria attuale

[343]:
```
hbd_complete.info(verbose=False, memory_usage="deep")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 119390 entries, 0 to 119389
Columns: 32 entries, hotel to reservation_status_date
dtypes: category(11), float64(4), int64(16), object(1)
memory usage: 27.2 MB
```

- Lo spazio occupato in memoria si è circa dimezzato due volte
- Come accennato il nostro obiettivo è realizzare un modello di classificazione per una specificia struttura ospitante, la struttura *City Hotel*
  - Il dataset prevede istanze di due strutture: *Resort Hotel* e *City Hotel*

[344]:
```
hbd_complete["hotel"].unique()
```

[344]:
```
[Resort Hotel, City Hotel]
Categories (2, object): [Resort Hotel, City Hotel]
```

- La preparazione dei dati viene effettuata sul dataset completo per poi rimuovere le istanze di *Resort Hotel* al termine
  - Il dataset completo ci sarà utile in seguito
- Visualizziamo il numero di valori distinti per ciascuna feature

[345]:
```
hbd_complete.nunique()
```

[345]:
```
hotel                               2
is_canceled                         2
lead_time                         479
arrival_date_year                   3
arrival_date_month                 12
arrival_date_week_number           53
arrival_date_day_of_month          31
stays_in_weekend_nights            17
stays_in_week_nights               35
adults                             14
children                            5
babies                              5
meal                                5
country                           177
market_segment                      8
distribution_channel                5
is_repeated_guest                   2
previous_cancellations             15
previous_bookings_not_canceled     73
reserved_room_type                 10
assigned_room_type                 12
booking_changes                    21
```

```
deposit_type                     3
agent                          333
company                        352
days_in_waiting_list           128
customer_type                    4
adr                           8879
required_car_parking_spaces      5
total_of_special_requests        6
reservation_status               3
reservation_status_date        926
dtype: int64
```

- Tutte le feature presentano una variabilità adatta alla loro semantica
- La variabili temporali associate all' arrivo del cliente in hotel attualmente presenti possono essere sostituite con altre variabili che presentano maggiore correlazione con ciò che stiamo cercando di prevedere
- Le variabili attualmente in nostro possesso sono:
  - `arrival_date_year`
  - `arrival_date_month`
  - `arrival_date_week_number`
  - `arrival_date_day_of_month`
- Per prima cosa vogliamo aggiungere il giorno della settimana (Lunedi, Martedi, ...)
  - In questo contesto è infatti più rilevante il giorno della settimana piuttosto che il giorno del mese
    - Per farlo è necessario avere a nostra disposizione un dizionario che ci consenta di ottenere, dato il nome del mese, il suo indice (1. Gennaio, 2. Febbrario, ...)

[346]:
```
import calendar
dict_month_convertion = dict((v,k) for k,v in enumerate(calendar.month_name))
```

- Mostriamo un esempio

[347]:
```
hbd_complete["arrival_date_month"].head(1)
```

[347]:
```
0    July
Name: arrival_date_month, dtype: category
Categories (12, object): [April, August, December, February, …, May, November,
October, September]
```

[348]:
```
hbd_complete["arrival_date_month"].head(1).map(dict_month_convertion)
```

[348]:
```
0    7
Name: arrival_date_month, dtype: category
Categories (12, int64): [4, 8, 12, 2, …, 5, 11, 10, 9]
```

- Utilizziamo questo dizionario per ottenere una serie che mappa, con il metodo appena mostrato, la variabile `arrival_date_month`

[349]:
```
arrival_date_month_number = hbd_complete["arrival_date_month"].
↪map(dict_month_convertion)
```

- Definiamo una funzione che presa in input una data (una stringa nel formato *giorno mese anno*) ci restituisca il nome del giorno

[350]:
```
import datetime

def findDay(date):
    day = datetime.datetime.strptime(date, '%d %m %Y').weekday()
    return (calendar.day_name[day])
```

- Realizziamo la nuova variabile `arrival_date_day`
  - Anche essa categorica

[351]:
```
arrival_date_day = []
for index, row in hbd_complete.iterrows():
    arrival_date_day.append(findDay("{0} {1} {2}".
↪format(row["arrival_date_day_of_month"], arrival_date_month_number[index],␣
↪row["arrival_date_year"])))

hbd_complete.insert(2, "arrival_date_day", arrival_date_day)
hbd_complete["arrival_date_day"] = hbd_complete["arrival_date_day"].
↪astype("category")
```

- Eliminiamo la variabile `arrival_date_day_of_month`, ovvero il giorno del mese di arrivo in hotel
- Eliminiamo anche la variabile `arrival_date_year`, in quanto non particolarmente utile

[352]:
```
hbd_complete.drop(inplace=True, axis=1, labels=['arrival_date_year',␣
↪'arrival_date_day_of_month'])
```

- La variabile `assigned_room_type` non è disponibile al momento della prenotazione, ma solo al momento del Check-In. Rappresenta la camera che viene assegnata al cliente al momento del suo arrivo in hotel, e non la camera prenotata.
  - E' dunque rimossa

[353]:
```
hbd_complete.drop(inplace=True, axis=1, labels=['assigned_room_type'])
```

- Osservando la descrizione delle variabili è possibile notare come vi sia una grossa affinità tra le variabili `is_canceled` e `reservation_status`
- I possibili valori di `reservation_status` sono...

[354]:
```
hbd_complete["reservation_status"].unique()
```

[354]:
```
[Check-Out, Canceled, No-Show]
Categories (3, object): [Check-Out, Canceled, No-Show]
```

- Il valore *Check-Out* potrebbe corrispondere alla mancata cancellazione, mentre il valore *Canceled* alla effettiva cancellazione. Anche il campo *No-Show* potrebbe essere considerato come prenotazione cancellata.
  - Definiziamo un dizionario in cui mappiamo i valori di `reservation_status`
    - Check-Out come non cancellata (0)
    - No-Show come cancellata (1)
    - Canceled come cancellata (1)

[355]:
```
mapping_reservation_status = {}
mapping_reservation_status['Check-Out'] = 0
mapping_reservation_status['No-Show'] = 1
mapping_reservation_status['Canceled'] = 1
```

- Mappiamo la variabile `reservation_status` con il dizionario creato

[356]:
```
reservation_status_mapped = hbd_complete["reservation_status"].
↪map(mapping_reservation_status)
```

- Verifichiamo se la serie ottenuta coincice con la variabile `is_canceled`.

[357]:
```
all(hbd_complete["is_canceled"] == reservation_status_mapped)
```

[357]:
```
True
```

- La considerazione era corretta, le variabili `is_canceled` e `reservation_status` coincidono.
  - Il valore *Check-Out* viene utilizzato quando la prenotazione non è stata cancellata
  - I valori *Canceled* e *No-Shown* vengono invece utilizzati quando la prenotazione è stata cancellata.
- Questa variabile va dunque rimossa in quanto coincide con la variabile da predire
  - Rimuoviamo anche `reservation_status_date`, in quanto inutile senza `reservation_status`

[358]:
```
hbd_complete.drop(inplace=True, axis=1, labels=['reservation_status',␣
↪'reservation_status_date'])
```

- Verifichiamo la presenza del valore `nan` nelle istanze

[359]:
```
hbd_complete.isnull().sum()
```

[359]:
```
hotel                           0
is_canceled                     0
arrival_date_day                0
lead_time                       0
arrival_date_month              0
arrival_date_week_number        0
stays_in_weekend_nights         0
```

```
stays_in_week_nights                0
adults                              0
children                            4
babies                              0
meal                                0
country                           488
market_segment                      0
distribution_channel                0
is_repeated_guest                   0
previous_cancellations              0
previous_bookings_not_canceled      0
reserved_room_type                  0
booking_changes                     0
deposit_type                        0
agent                           16340
company                        112593
days_in_waiting_list                0
customer_type                       0
adr                                 0
required_car_parking_spaces         0
total_of_special_requests           0
dtype: int64
```

- Il valore `nan` è presente nelle variabili:
  - `children`
  - `country`
  - `agent`
  - `company`
- Per le variabili `children` e `country` tale valore non è accettabile, per cui rimuovo quelle istanze

[360]:
```
hbd_complete.dropna(subset=["country", "children"], inplace=True)
```

- La variabile `company` è nulla in quasi tutte le istanze, mentre `agent` per una buona parte di esse.
- Il valore nullo in questo caso è di nostro interesse in quanto implica che per quella prenotazione non è presente, rispettivamente, una company e/o un agent
  - Sostituiamo i valori di queste due variabili con 0 e 1
    - 0 se il valore è `nan`
    - 1 altrimenti
  - In questo modo manteniamo l'informazione relativa alla presenta (o assenza) di un agent e/o di una company in una prenotazione
    - Non è di particolare interesse sapere esattamente chi sia il soggetto interessato

[361]:
```
hbd_complete.loc[hbd_complete["agent"].isnull(), "agent"] = 0
hbd_complete.loc[hbd_complete["agent"] != 0, "agent"] = 1
hbd_complete.loc[hbd_complete["company"].isnull(), "company"] = 0
```

```
hbd_complete.loc[hbd_complete["company"] != 0, "company"] = 1
```

- Un' alternativa era quella di selezionare solo parte dei possibili valori di `agent` e `company` (i più frequenti) e considerare questa variabile come categorica. Ai valori più frequenti ottenuti ne andrebbero aggiunti due: *Nessuno* per sostituire tutti i valori `nan` e *Altro* , per sostituire tutti i valori diversi da `nan` non presenti fra i valori più frequenti.
  - Si è optato di non seguire questa strada
- Non sono più presenti valori `nan`

[362]:
```
hbd_complete.isnull().sum()
```

[362]:
```
hotel                           0
is_canceled                     0
arrival_date_day                0
lead_time                       0
arrival_date_month              0
arrival_date_week_number        0
stays_in_weekend_nights         0
stays_in_week_nights            0
adults                          0
children                        0
babies                          0
meal                            0
country                         0
market_segment                  0
distribution_channel            0
is_repeated_guest               0
previous_cancellations          0
previous_bookings_not_canceled  0
reserved_room_type              0
booking_changes                 0
deposit_type                    0
agent                           0
company                         0
days_in_waiting_list            0
customer_type                   0
adr                             0
required_car_parking_spaces     0
total_of_special_requests       0
dtype: int64
```

- Verifichiamo la tipologia delle variabili in nostro possesso

[363]:
```
hbd_complete.dtypes
```

[363]:
```
hotel                           category
is_canceled                        int64
```

```
arrival_date_day                  category
lead_time                            int64
arrival_date_month                category
arrival_date_week_number             int64
stays_in_weekend_nights              int64
stays_in_week_nights                 int64
adults                               int64
children                           float64
babies                               int64
meal                              category
country                           category
market_segment                    category
distribution_channel              category
is_repeated_guest                    int64
previous_cancellations               int64
previous_bookings_not_canceled       int64
reserved_room_type                category
booking_changes                      int64
deposit_type                      category
agent                              float64
company                            float64
days_in_waiting_list                 int64
customer_type                     category
adr                                float64
required_car_parking_spaces          int64
total_of_special_requests            int64
dtype: object
```

- Converto ad intero le variabili
  - children
  - agent
  - company

[364]:
```
hbd_complete["agent"] = hbd_complete["agent"].astype("int64")
hbd_complete["company"] = hbd_complete["company"].astype("int64")
hbd_complete["children"] = hbd_complete["children"].astype("int64")
```

[365]:
```
print("children: ", hbd_complete["children"].dtype)
print("agent: ", hbd_complete["children"].dtype)
print("company: ", hbd_complete["children"].dtype)
```

```
children:  int64
agent:  int64
company:  int64
```

- Abbiamo terminato la preparazione dei dati
  - Viene ora creato un nuovo dataframe senza le istanze di *Resort Hotel*

[366]:
```
hbd = hbd_complete[hbd_complete["hotel"] == "City Hotel"].copy()
hbd["hotel"].unique()
```

[366]:
```
[City Hotel]
Categories (1, object): [City Hotel]
```

- Il dataset ora presenta...

[367]:
```
print(hbd.shape[0], "istanze")
```

```
79302 istanze
```

- La variabile `hotel` è diventata inutile, è quindi rimossa

[368]:
```
hbd.drop(inplace=True, axis=1, labels=['hotel'])
```

1.3 Analisi esplorativa

- Visualizziamo le statistiche principali (media, dev, standard, ...) delle variabili

[369]:
```
hbd.describe().T
```

[369]:

	count	mean	std	min	25%	\
is_canceled	79302.0	0.417089	0.493081	0.0	0.0	
lead_time	79302.0	109.740183	110.953223	0.0	23.0	
arrival_date_week_number	79302.0	27.173564	13.397803	1.0	17.0	
stays_in_weekend_nights	79302.0	0.795339	0.884985	0.0	0.0	
stays_in_week_nights	79302.0	2.182896	1.456096	0.0	1.0	
adults	79302.0	1.851126	0.509013	0.0	2.0	
children	79302.0	0.091397	0.372230	0.0	0.0	
babies	79302.0	0.004943	0.084338	0.0	0.0	
is_repeated_guest	79302.0	0.025624	0.158010	0.0	0.0	
previous_cancellations	79302.0	0.079771	0.415543	0.0	0.0	
previous_bookings_not_canceled	79302.0	0.132418	1.693708	0.0	0.0	
booking_changes	79302.0	0.187435	0.608718	0.0	0.0	
agent	79302.0	0.897594	0.303183	0.0	1.0	
company	79302.0	0.046417	0.210389	0.0	0.0	
days_in_waiting_list	79302.0	3.227914	20.874486	0.0	0.0	
adr	79302.0	105.326470	43.590608	0.0	79.2	
required_car_parking_spaces	79302.0	0.024375	0.154946	0.0	0.0	
total_of_special_requests	79302.0	0.547035	0.780835	0.0	0.0	

	50%	75%	max
is_canceled	0.0	1.0	1.0
lead_time	74.0	163.0	629.0
arrival_date_week_number	27.0	38.0	53.0
stays_in_weekend_nights	1.0	2.0	16.0
stays_in_week_nights	2.0	3.0	41.0

```
adults                            2.0    2.0     4.0
children                          0.0    0.0     3.0
babies                            0.0    0.0    10.0
is_repeated_guest                 0.0    0.0     1.0
previous_cancellations            0.0    0.0    21.0
previous_bookings_not_canceled    0.0    0.0    72.0
booking_changes                   0.0    0.0    21.0
agent                             1.0    1.0     1.0
company                           0.0    0.0     1.0
days_in_waiting_list              0.0    0.0   391.0
adr                              99.9  126.0  5400.0
required_car_parking_spaces       0.0    0.0     3.0
total_of_special_requests         0.0    1.0     5.0
```

- Tutte le variabili presentano come minimo il valore (~) 0
- I valori non presentano la stessa scala, la standardizzazione potrà quindi essere certamente utile
- Alcune variabili mostrano un valore di massimo che in relazione alla loro semantica pare poco sensato
  - Visualizziamo il box plot di queste variabili

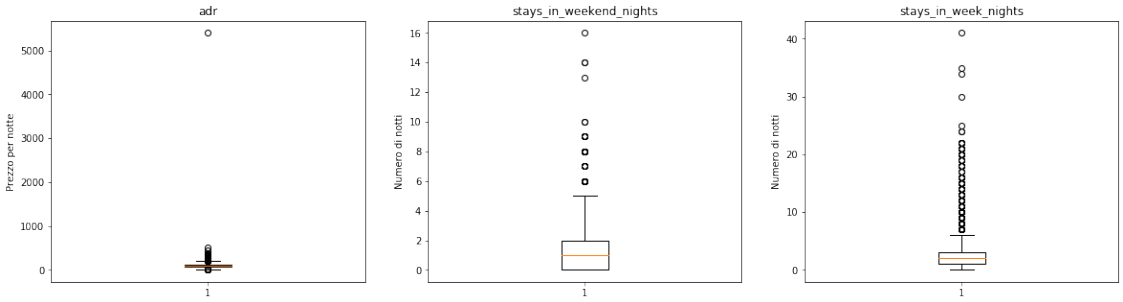
[370]:
```
plt.figure(figsize=(20, 5))

plt.subplot(1, 3, 1)
plt.title('adr')
plt.boxplot(hbd['adr'])
plt.ylabel('Prezzo per notte')

plt.subplot(1, 3, 2)
plt.title('stays_in_weekend_nights')
plt.boxplot(hbd['stays_in_weekend_nights'])
plt.ylabel('Numero di notti')

plt.subplot(1, 3, 3)
plt.title('stays_in_week_nights')
plt.boxplot(hbd['stays_in_week_nights'])
plt.ylabel('Numero di notti')

plt.show()
```



- Possiamo notare alcuni outliers in tutte e tre le variabili
  - Il valore di massimo della variabile `adr` è estremamente lontanto dalla media
    - Viene rimosso in quanto potrebbe potenzialmente compromette i grafici dell'analisi esplorativa

[371]:
```
hbd = hbd[hbd["adr"] != hbd["adr"].max()]
```

- In un problema di classificazione è utile visualizzare quanto le variabili siano correlate tra loro
  - Calcoliamo la correlazione Pearson tra tutte le coppie di features

[372]:
```
hbd.corr().style.background_gradient(cmap='Spectral').set_precision(2)
```

[372]:
```
<pandas.io.formats.style.Styler at 0x268875479c8>
```

- A noi interessa particolamente la prima colonna, ovvero la relazione che intercorre tra la variabile dipendendente e tutte le altre
  - Prendiamo questi valori in valore assoluto e in ordine decrescente

[373]:
```
cancel_corr = hbd.corr()["is_canceled"]
cancel_corr.abs().sort_values(ascending=False)[1:]
```

[373]:
```
lead_time                        0.309369
total_of_special_requests        0.293807
previous_cancellations           0.166758
booking_changes                  0.149448
required_car_parking_spaces      0.133070
company                          0.092296
agent                            0.066481
is_repeated_guest                0.065795
days_in_waiting_list             0.061039
adults                           0.053412
previous_bookings_not_canceled   0.053117
stays_in_week_nights             0.048691
babies                           0.030171
children                         0.027002
adr                              0.014641
stays_in_weekend_nights          0.007085
```

```
arrival_date_week_number          0.001331
Name: is_canceled, dtype: float64
```

- Apprendiamo che le variabili numeriche più correlate con la variabile dipendente sono:
  - `lead_time`
  - `total_of_special_requests`
  - `required_car_parking_spaces`
  - `previous_cancellations`
  - `booking_changes`
- Visualizziamo anche i valori più alti fra tutti quelli ottenuti (sempre in valore assoluto e sempre in ordine decrescente)
  - Sono indice di collinearità, è quindi opportuno tenerne conto
  - La matrice con i valori di correlazione è simmetrica, estraiamo dunque solo metà di essa

[374]:
```
(hbd.corr().abs().where(np.triu(np.ones(cancel_corr.abs().shape), k=1).
 ↪astype(np.bool))
                 .stack()
                 .sort_values(ascending=False)
                 .head(10))
```

[374]:
```
agent                           company                         0.643897
is_repeated_guest               previous_bookings_not_canceled  0.451963
                                company                         0.395515
previous_cancellations          previous_bookings_not_canceled  0.392139
children                        adr                             0.346216
is_canceled                     lead_time                       0.309369
previous_bookings_not_canceled  company                         0.304077
is_canceled                     total_of_special_requests       0.293807
is_repeated_guest               agent                           0.291368
adults                          adr                             0.290379
dtype: float64
```

- Possiamo per esempio notare che…

[375]:
```
hbd.groupby(['is_repeated_guest','previous_bookings_not_canceled']).size().
 ↪unstack().fillna(0)[[0, 1, 2, 3, 4, 5]]
```

[375]:
```
previous_bookings_not_canceled       0      1      2      3     4     5
is_repeated_guest
0                                77096.0  136.0    5.0    2.0   6.0   0.0
1                                  617.0  433.0  187.0  127.0  96.0  90.0
```

- La forte correlazione tra le variabili `is_repeated_guest` e `previous_bookings_not_canceled` è dovuta al fatto che quando un cliente è nuovo (la gran parte dei casi) ovviamente non ha a suo nome nessuna prenotazione cancellata
- Oppure che…

[376]:
```
hbd.groupby(['previous_cancellations', 'previous_bookings_not_canceled']).
↪size().unstack().fillna(0)[[0,1,2]].head(3)
```

[376]:
```
previous_bookings_not_canceled        0      1      2
previous_cancellations
0                                 72759.0  526.0  173.0
1                                  4936.0   25.0   16.0
2                                     4.0    5.0    2.0
```

- L'osservazione enunciata sopra può essere applicata anche alle variabili `previous_cancellations` e `previous_bookings_not_canceled`
  - Quando un cliente è nuovo non ha a suo nome nessuna prenotazione, sia essa cancellata o non cancellata
- Visualizziamo i grafici a dispersione delle variabili che, tenuto conto del dominio analizzato, possono essere ritenute le più interessanti

[377]:
```
import seaborn as sns
hbd_temp = hbd.copy()
hbd_temp["is_canceled"] = hbd_temp["is_canceled"].map(lambda value: "N" if␣
↪value is 0 else "Y")
sns.pairplot(hbd_temp[['previous_cancellations',␣
↪'previous_bookings_not_canceled', 'total_of_special_requests',␣
↪'required_car_parking_spaces', 'booking_changes', 'is_canceled']],␣
↪hue="is_canceled", palette="husl", diag_kind="hist")
```

[377]:
```
<seaborn.axisgrid.PairGrid at 0x268894e8808>
```

- Osserviamo che…
  - Quando la variabile `required_car_parking_spaces` assume valore maggiore o uguale a 1 nessuna prenotazione risulta essere cancellata
  - Quando vengono effettuate molte modifiche alla prenotazione in genere il cliente non ha mai effettuato ne cancellazioni ne altro.
    - Possiamo supporre che quando vengono effettuate molte modifiche alla prenotazione questa provenga da un nuovo cliente
  - Le prenotazioni che prevedono molte modifiche in genere non vengono cancellate
  - All'aumentare del numero di prenotazioni non cancellate si dimostrano meno presenti prenotazioni cancellate
- Analiziamo alcune variabili più nel dettaglio, iniziando dalla variabile dipendente

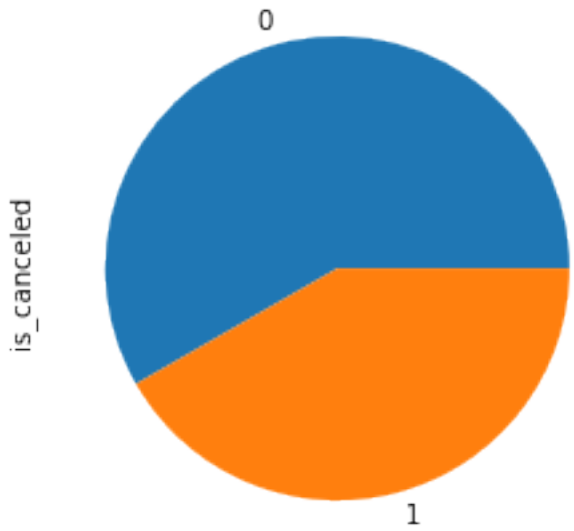
1.3.1 Variabile `is_canceled`

- Stampiamo il numero di valori `0` e `1` della variabile `is_canceled`, e rappresentiamo la distribuzione di tali valori in un diagramma a torta

[378]:
```
hbd["is_canceled"].value_counts()
```

[378]:
```
0    46226
1    33075
Name: is_canceled, dtype: int64
```

[379]:
```
hbd["is_canceled"].value_counts().plot.pie();
```



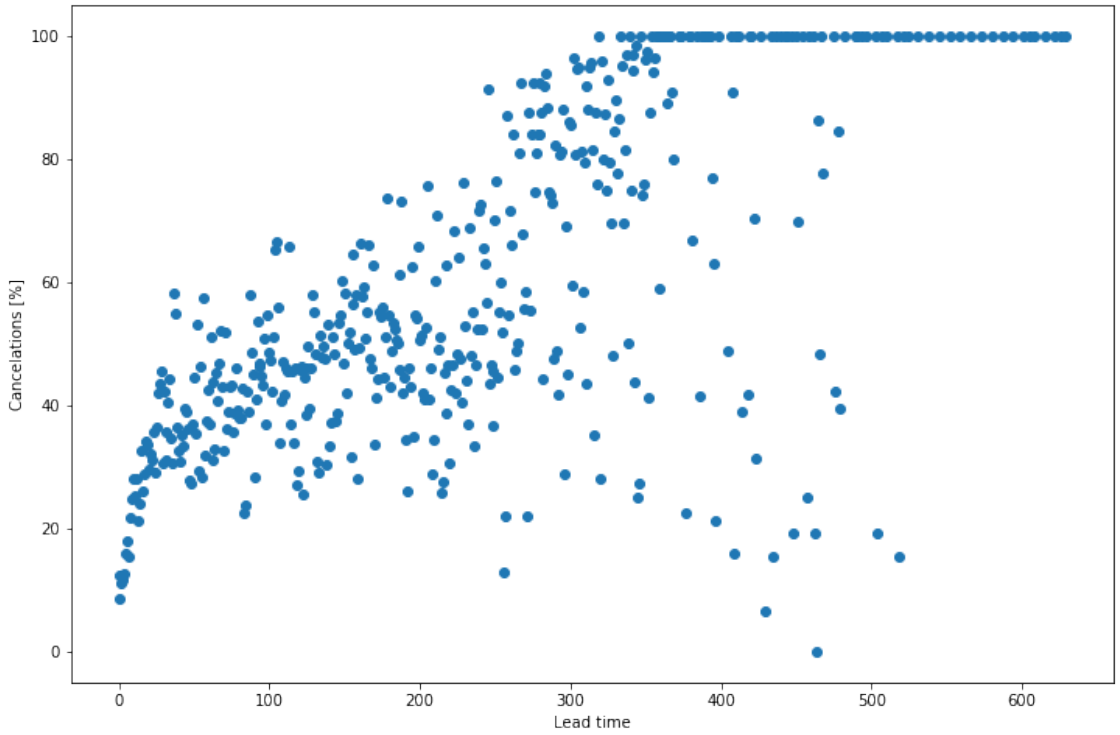
- La suddivisione delle istanze nelle classi è abbastanza bilanciata, non siamo dunque soggetti ai problemi che un forte sblinciamento comporterebbe.
- Analizziamo ora le variabili che hanno ottenuto il maggiore valore di correlazione con la variabile dipendende, ovvero:
  - `lead_time`
  - `total_of_special_requests`
  - `required_car_parking_spaces`
  - `previous_cancellations`
  - `booking_changes`

1.3.2 Variabile `lead_time`

- Visualizziamo la percentuale di cancellazione per i valori di `lead_time`

[380]:
```
plt.figure(figsize=(12, 8))
lead_time_describe = hbd.groupby("lead_time")["is_canceled"].describe()
plt.scatter(lead_time_describe.index, lead_time_describe["mean"] * 100)
plt.xlabel("Lead time")
plt.ylabel("Cancelations [%]")
```

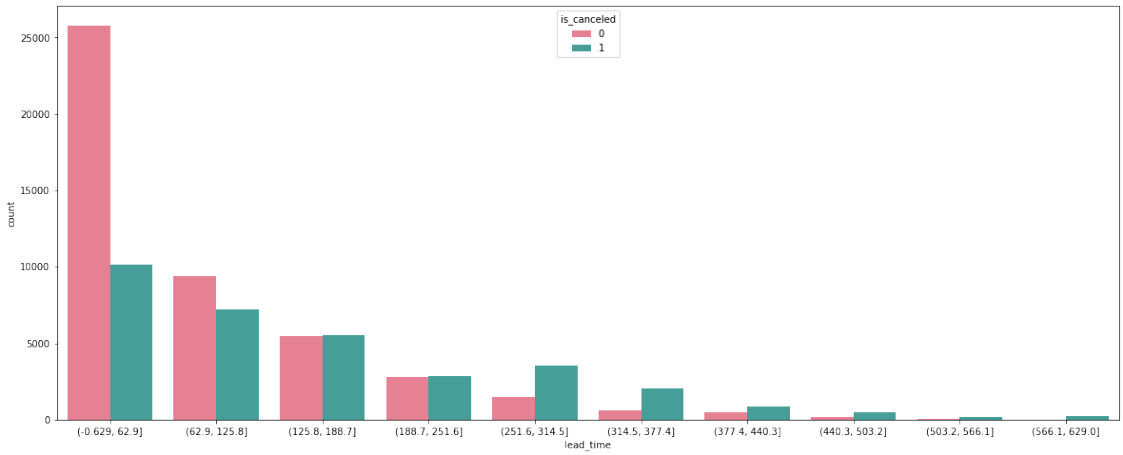
[380]: Text(0, 0.5, 'Cancelations [%]')



- Le prenotazione fatte qualche giorno prima dell'arrivo in struttura sono raramente cancellate, a differenza di prenotazioni fatte a distanza di mesi (se non di anni)
- Visualizziamo un l'istogramma in cui andiamo a quantificare la frequenza delle classi nei valori che la variabile assume

[381]:
```
from matplotlib import rcParams
rcParams['figure.figsize'] = 20, 8
sns.countplot(x=pd.cut(hbd["lead_time"], 10), hue='is_canceled', data = hbd,␣
↪palette="husl")
```

[381]: <matplotlib.axes._subplots.AxesSubplot at 0x268878f13c8>



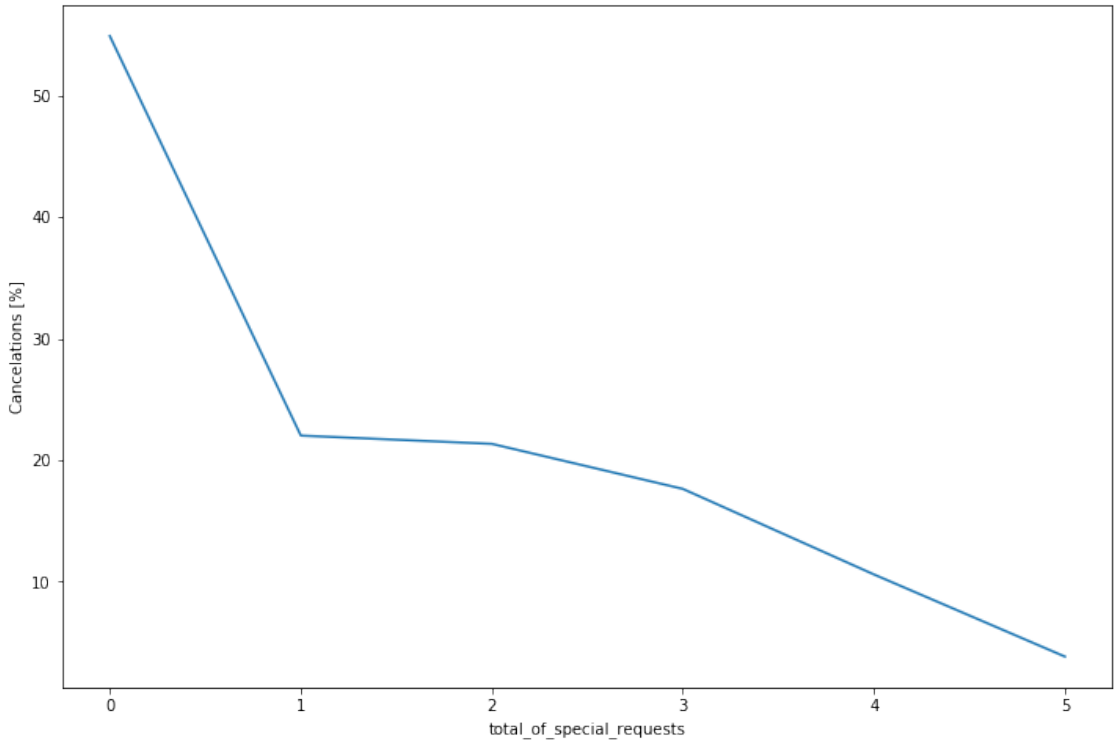
- Le prenotazioni più frequenti sono quelle a breve termine, ovvero senza lunghi periodi fra il momento della prenotazione e l'arrivo in hotel
- Medesima cosa vale per le cancellazioni, anche esse maggiori nelle prenotazioni a breve termine

1.3.3 Variabile `total_of_special_requests`

- Visualizziamo la percentuale di cancellazione per i valori di `total_of_special_requests`

[382]:
```
plt.figure(figsize=(12, 8))
plt.xlabel("Total of special request")
plt.ylabel("Cancelations [%]")
(100 * (hbd.groupby("total_of_special_requests").sum()["is_canceled"] /␣
 ↪hbd["total_of_special_requests"].value_counts())).plot()
```

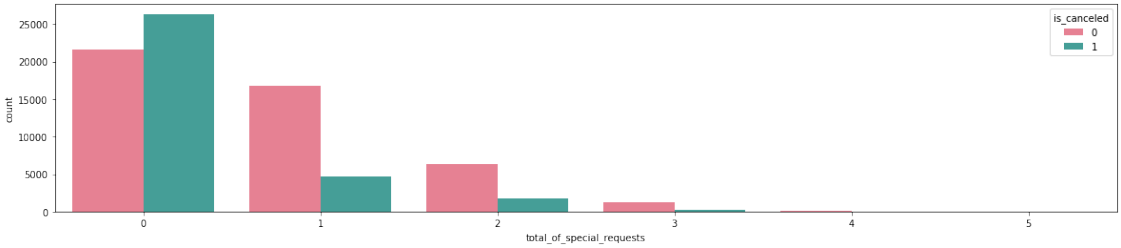
[382]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x268878ef088>
```



- Visualizziamo un l'istogramma in cui andiamo a quantificare la frequenza delle classi nei valori che la variabile assume

[383]:
```
from matplotlib import rcParams
rcParams['figure.figsize'] = 20, 4
sns.countplot(x="total_of_special_requests", hue='is_canceled', data = hbd,␣
↪palette="husl")
```

[383]: <matplotlib.axes._subplots.AxesSubplot at 0x268878e1848>



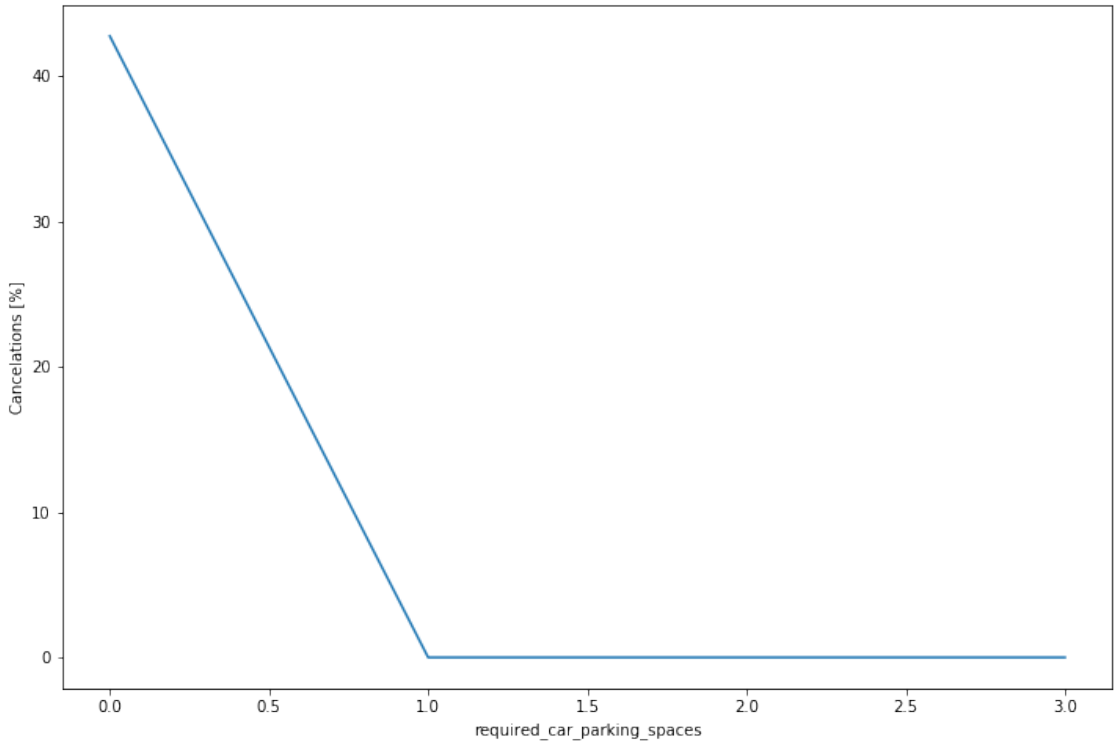
- All'aumentare del numero di richieste speciali la percentuale di cancellazioni diminuisce
  - Tanto più un cliente effettua richieste speciali tanto più è raro che cancelli la sua prenotazione

1.3.4 Variabile `required_car_parking_spaces`

- Visualizziamo la percentuale di cancellazione per i valori di `required_car_parking_spaces`

[384]:
```
plt.figure(figsize=(12, 8))
plt.xlabel("Required car parking spaces")
plt.ylabel("Cancelations [%]")
(100 * (hbd.groupby("required_car_parking_spaces").sum()["is_canceled"] /␣
 ↪hbd["required_car_parking_spaces"].value_counts())).plot()
```

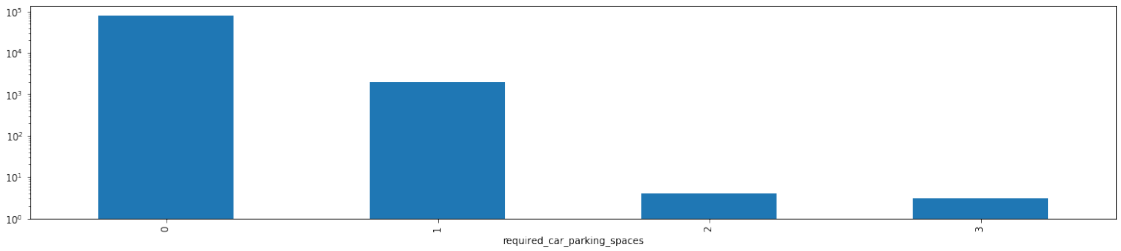
[384]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2688d9d7788>
```



- Visualizziammo anche il numero di prenotazioni effettuate per ciacun valore di `required_car_parking_spaces`

[385]:
```
hbd.groupby("required_car_parking_spaces").size().plot.bar(log=True)
```

[385]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x268f037ea88>
```



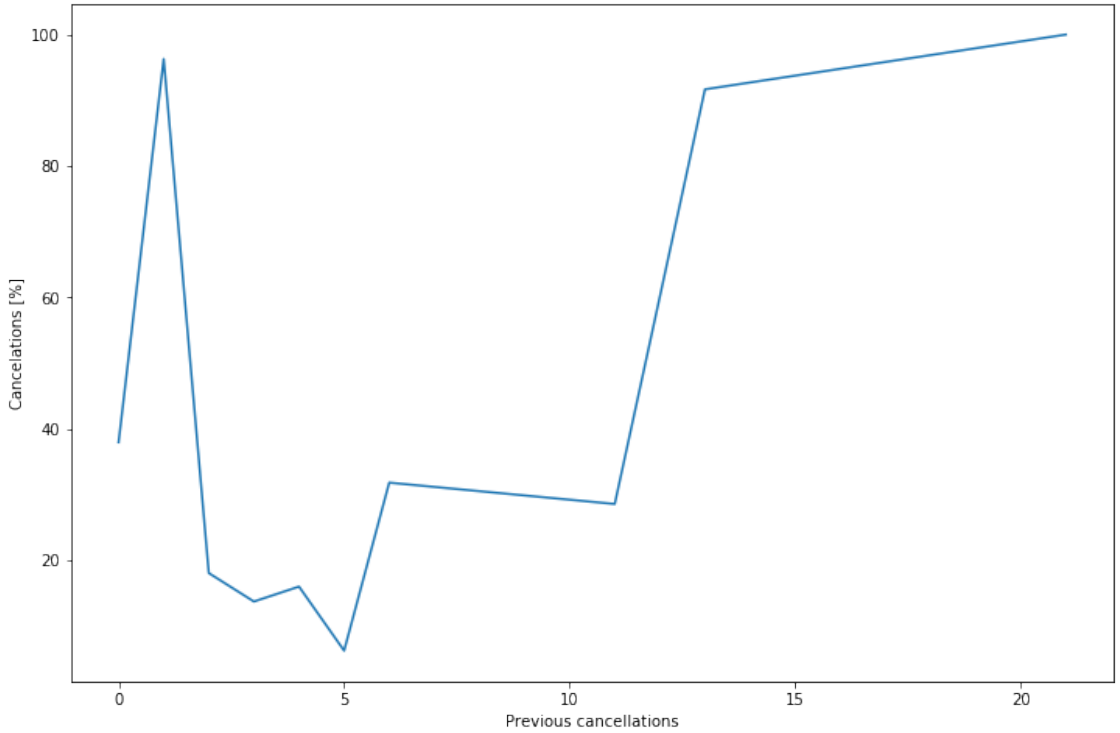
- Nessuna prenotazione per la quale era previsto almeno un posto macchina è mai stata cancellata
  - Chi viene in macchina in genere non cancella la propria prenotazione

1.3.5 Variabile `previous_cancellations`

- Visualizziamo la percentuale di cancellazione per i valori di `previous_cancellations`

[386]:
```
plt.figure(figsize=(12, 8))
plt.xlabel("Previous cancellations")
plt.ylabel("Cancelations [%]")
(100 * (hbd.groupby("previous_cancellations").sum()["is_canceled"] /␣
 ↪hbd["previous_cancellations"].value_counts())).plot()
```

[386]: `<matplotlib.axes._subplots.AxesSubplot at 0x268a2cc59c8>`



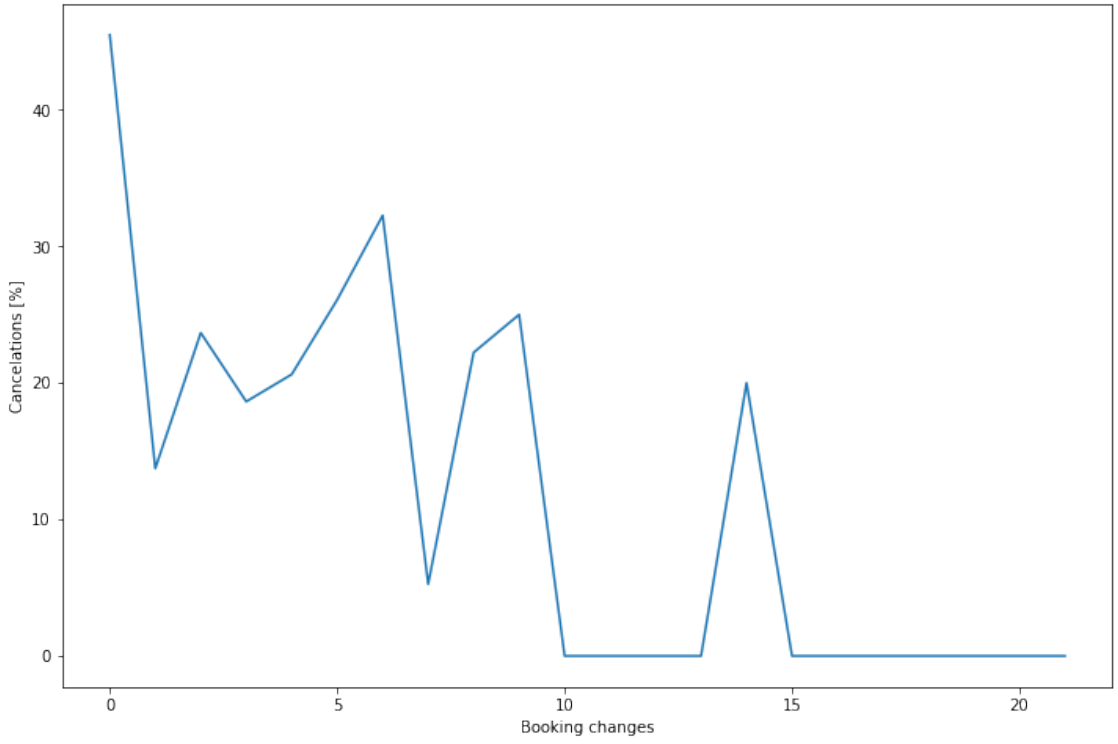
- Eccetto inizialmente, all'aumentare del numero di prenotazioni già cancellate sembra essere sempre più frequente un'ulteriore cancellazione
  - Chi ha già cancellato molte prenotazioni probabilmente lo rifarà (tanto più quante sono le cancellazioni fatte)

1.3.6 Variabile `booking_changes`

- Visualizziamo la percentuale di cancellazione per i valori di `booking_changes`

[387]:
```
plt.figure(figsize=(12, 8))
plt.xlabel("Booking changes")
plt.ylabel("Cancelations [%]")
(100 * (hbd.groupby("booking_changes").sum()["is_canceled"] /␣
 ↪hbd["booking_changes"].value_counts())).plot()
```

[387]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x268a3159b88>
```

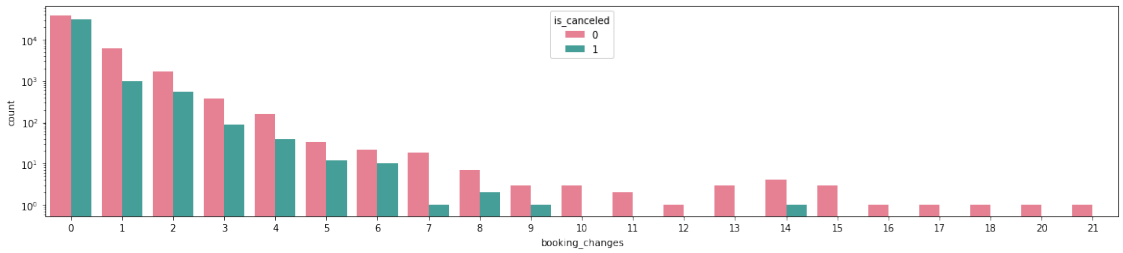


- Visualizziamo un l'istogramma in cui andiamo a quantificare la frequenza delle classi nei valori che la variabile assume

[388]:
```
from matplotlib import rcParams
rcParams['figure.figsize'] = 20, 4
```

```
sns.countplot(x="booking_changes", hue='is_canceled', data = hbd,␣
↪palette="husl", log=True)
```

[388]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x268a2bd1e08>
```



- All'aumentare del numero di richieste di cambiamento la percentuale di cancellazioni tende a diminuire
  - Tanti più cambiamenti un cliente richiede, tanto più è raro che cancelli la sua prenotazione
- Mostriamo ora la variabile `country`

1.3.7 Variabile `country`

- Valutiamo, per ciascuno stato, il numero di prenotazioni e cancellazioni

[389]:
```
cancellation_by_state = hbd.groupby(['country']).sum()["is_canceled"]
reservation_by_state = hbd.groupby(['country']).size()
```

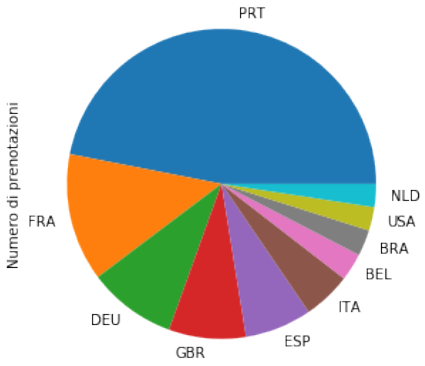
- Memorizziamo i primi 10 stati per numero di prenotazioni

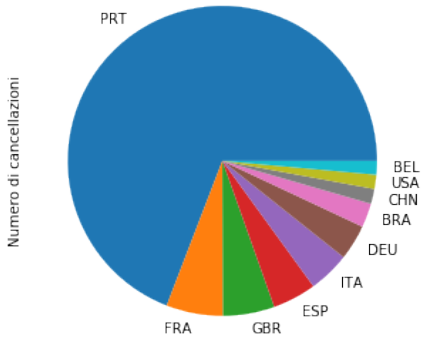
[390]:
```
bigger_states = reservation_by_state.sort_values(ascending=False).head(10)
```

- Visualizziamo due grafici a torta in cui vengono mostrati:
  - Gli stati con più prenotazioni
  - Gli stati con più cancellazioni

[391]:
```
fig = plt.figure(figsize=(16, 5))
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122)
reservation_by_state.sort_values(ascending=False).head(10).plot.pie(ax=ax1)
cancellation_by_state.sort_values(ascending=False).head(10).plot.pie(ax=ax2)
ax1.set(ylabel="Numero di prenotazioni")
ax2.set(ylabel="Numero di cancellazioni")
```

[391]:
```
[Text(0, 0.5, 'Numero di cancellazioni')]
```





- Lo stato che predomica maggiormente è *PRT* , ovvero il Portogallo
- Poichè l'hotel è situato in Portogallo è un risultato tutto sommato scontato

- Visualizziamo un grafico a barre con la percentuale di cancellazione relativa agli stati
  - Indichiamo colorando in modo diverso gli stati che risultano essere presenti nell'insieme `bigger_states`, ovvero i primi 10 stati per numero di prenotazioni
- Calcoliamo dunque la percentuale di cancellazioni di tutti gli statti

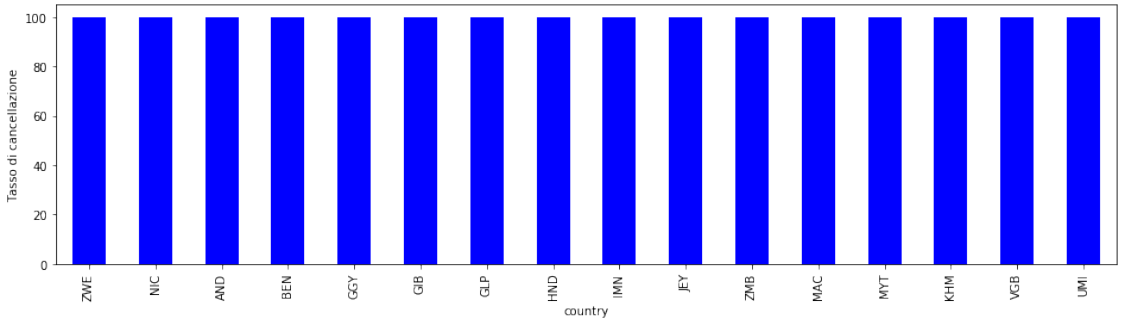
[392]:
```
ratio_cancellation_by_state = (100* cancellation_by_state /␣
↪reservation_by_state).sort_values(ascending=False)
```

- Mostraimo prima l'insieme di stati che presenta percentuale di cancellazione pari al 100%
  - Definisco i colori delle barre in funzione della presenza o meno dello stato in `bigger_states`

[393]:
```
state100 = ratio_cancellation_by_state[ratio_cancellation_by_state==100]
condition = state100.index.isin(bigger_states.index)
colors = ""
for element in condition:
    colors += "b" if element == False else "r"
```

[394]:
```
state100.plot.bar(figsize=(16,4), color = list(colors))
plt.ylabel("Tasso di cancellazione")
```

[394]:
```
Text(0, 0.5, 'Tasso di cancellazione')
```

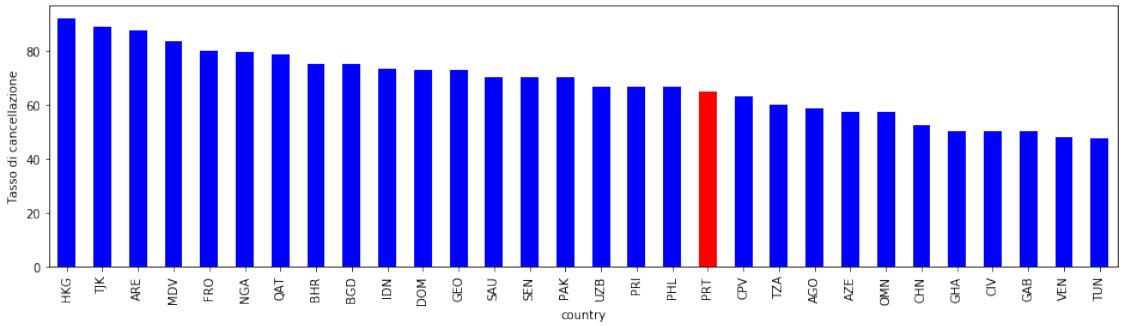


- Nessuno stato con un gran numero di prenotazioni ha percentuale di cancellazione pari a 100%
- Proviamo ora con i 30 stati successivi, replicando lo stesso approccio

[395]:
```
statenot100 = ratio_cancellation_by_state[ratio_cancellation_by_state!=100].
↪head(30)
condition = statenot100.index.isin(bigger_states.index)
colors = ""
for element in condition:
    colors += "b" if element == False else "r"
```

[396]:
```
statenot100.plot.bar(figsize=(16,4), color = list(colors))
plt.ylabel("Tasso di cancellazione")
```

[396]:
```
Text(0, 0.5, 'Tasso di cancellazione')
```



- In questo grafico troviamo il Portogallo (*PRT*), nonchè lo stato con maggior numero di prenotazioni (e maggior numero di cancellazioni)
- Mostriamo la percentuale esatta

[397]:
```
ratio_cancellation_by_state["PRT"]
```

[397]:
```
64.85543530931999
```

- I portoghesi tendono a cancellare le proprie prenotazioni molto frequentemente
- Andiamo ora ad analizzare la variabile `deposit_type` e tutte le variabili che trattano il periodo di arrivo e la permanenza in albergo, in particolare:
  - `stays_in_weekend_nights`
  - `stays_in_week_nights`
  - `arrival_date_day`
  - `arrival_date_month`,
  - `arrival_date_week_number`

1.3.8 Variabile `deposit_type`

- Visualizziamo due grafici a torta
  - Nel primo per ogni valore è mostrato il numero di prenotazioni
  - Nel secondo per ogni valore è mostrato il numero di cancellazioni

[398]:
```
fig = plt.figure(figsize=(16, 5))
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122)
hbd["deposit_type"].value_counts().plot.pie(ax=ax1)
hbd.groupby("deposit_type").sum()["is_canceled"].plot.pie(ax=ax2)
ax1.set(ylabel="Numero di prenotazioni")
ax2.set(ylabel="Numero di cancellazioni")
```

[398]:
```
[Text(0, 0.5, 'Numero di cancellazioni')]
```

- Il valore Refundable è praticamente inesistente

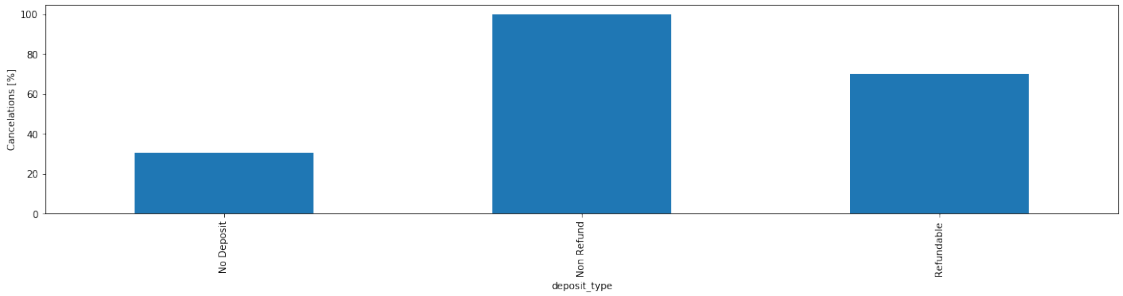
[399]:
```
hbd["deposit_type"].value_counts()
```

[399]:
```
No Deposit    66428
Non Refund    12853
Refundable       20
Name: deposit_type, dtype: int64
```

- Visualizziamo un grafico a barre di `deposit_type`, in cui in viene visualizzato per ciascun valore la percentuale di cancellazioni registrata

[400]:
```
(100 * hbd.groupby("deposit_type").sum()["is_canceled"] / hbd["deposit_type"].
↪value_counts()).plot.bar()
plt.ylabel("Cancelations [%]")
```

[400]:
```
Text(0, 0.5, 'Cancelations [%]')
```



- Visualizziamo i valori esatti

[401]:
```
(hbd.groupby("deposit_type").sum()["is_canceled"] / hbd["deposit_type"].
↪value_counts())
```

[401]:
```
deposit_type
No Deposit    0.304570
Non Refund    0.998133
Refundable    0.700000
dtype: float64
```

- Il grafico evidenzia che per il valore 'Non Refund' la percentuale di cancellazioni è pari quasi al 100%
- E' un po controintuitivo considerando il suo significato
  - Verifichiamo, per averne certezza, quante istanze con valore `Non Refund` sono state cancellate e quante no

[402]:
```
hbd[hbd["deposit_type"] == "Non Refund"]["is_canceled"].value_counts()
```

[402]:
```
1    12829
0       24
Name: is_canceled, dtype: int64
```

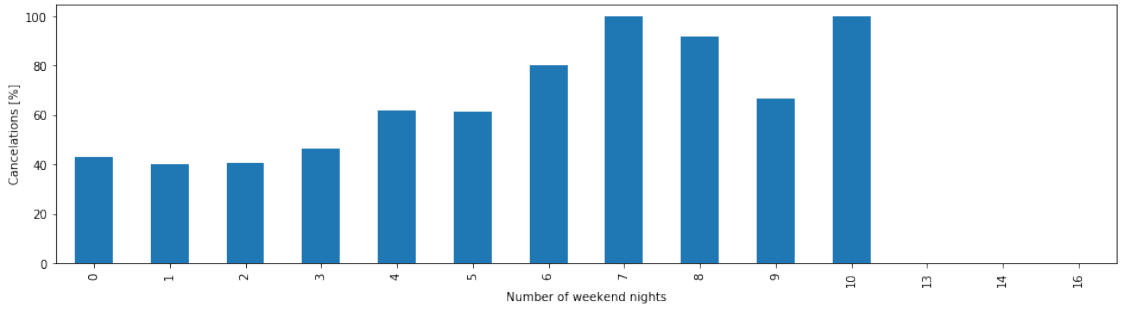
- Effettivamente pare che quasi tutte le prenotazioni *Non Refund* siano state cancellate
  - Poichè pare un fatto particolarmente strano, sarà opportuno verificare in modo attento quanto questa variabile incida sul risultato

1.3.9 Variabili `stays_in_weekend_nights`, `stays_in_week_nights`

- Mostriamo due grafici a barre in cui visualizziamo per ogni valore delle variabili `stays_in_weekend_nights` e `stays_in_week_nights`, ovvero il numero di notti prenotate rispettivamente durante il finesettimana e non durante il finesettimana, la percentuale di cancellazioni

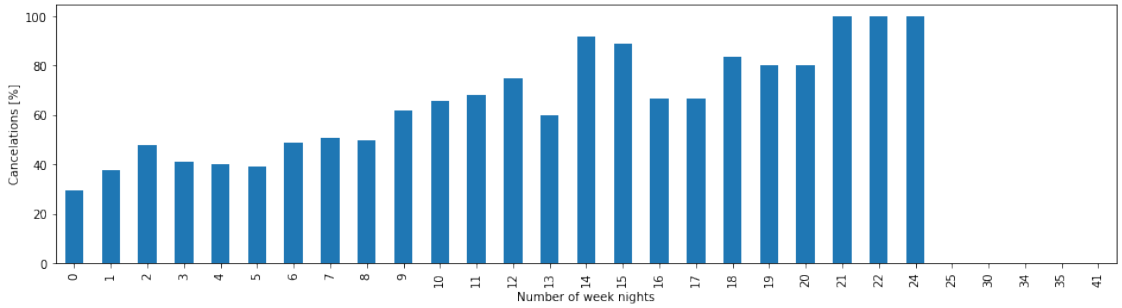
[403]:
```
(100 * hbd.groupby("stays_in_weekend_nights").sum()["is_canceled"] /␣
 ↪hbd["stays_in_weekend_nights"].value_counts()).plot.bar(figsize=(16,4))
plt.ylabel("Cancelations [%]")
plt.xlabel("Number of weekend nights")
```

[403]:
```
Text(0.5, 0, 'Number of weekend nights')
```



[404]:
```
(100 * hbd.groupby("stays_in_week_nights").sum()["is_canceled"] /␣
 ↪hbd["stays_in_week_nights"].value_counts()).plot.bar(figsize=(16,4))
plt.ylabel("Cancelations [%]")
plt.xlabel("Number of week nights")
```

[404]:
```
Text(0.5, 0, 'Number of week nights')
```



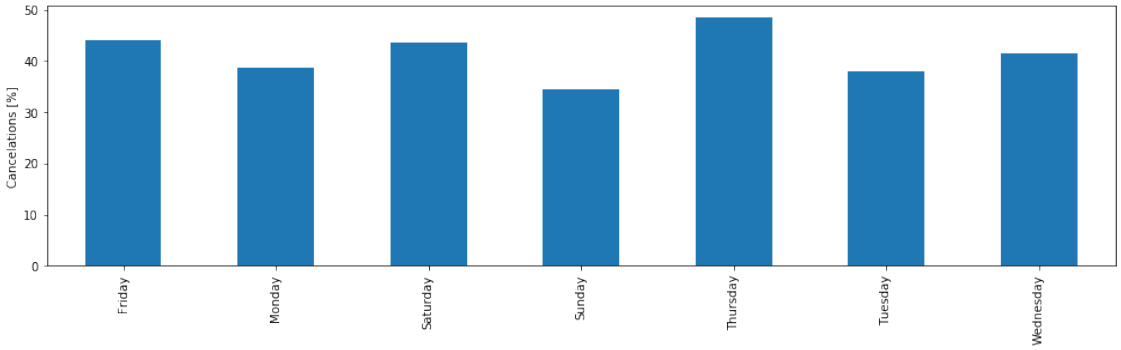
- In entrambi i casi possiamo notare all'aumentare del numero di notti prenotate i clienti tendono a cancellare le prenotazioni con più frequenza
  - Vengono raggiunti picchi pari al 100%

1.3.10 Variabili `arrival_date_day`, `arrival_date_month`, `arrival_date_week_number`

- Mostriamo le percentuali di cancellazione delle variabili `arrival_date_day`, `arrival_date_month`, `arrival_date_week_number`

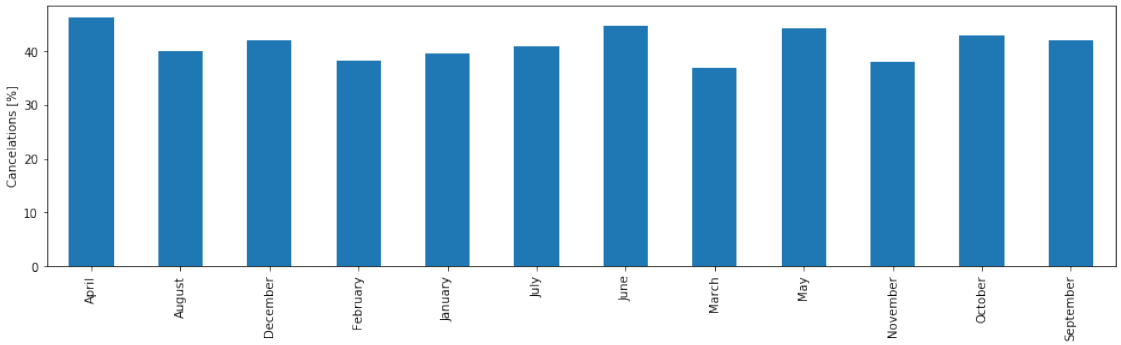
[405]:
```
(100 * hbd.groupby("arrival_date_day").sum()["is_canceled"] /␣
↪hbd["arrival_date_day"].value_counts()).plot.bar(figsize=(16,4))
plt.ylabel("Cancelations [%]")
```

[405]: Text(0, 0.5, 'Cancelations [%]')



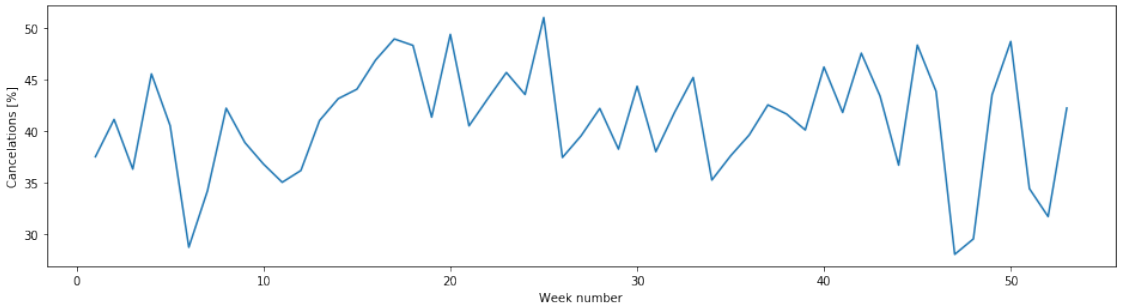
[406]:
```
(100 * hbd.groupby("arrival_date_month").sum()["is_canceled"] /␣
↪hbd["arrival_date_month"].value_counts()).plot.bar(figsize=(16,4))
plt.ylabel("Cancelations [%]")
```

[406]: Text(0, 0.5, 'Cancelations [%]')



[407]:
```
(100 * hbd.groupby("arrival_date_week_number").sum()["is_canceled"] /␣
↪hbd["arrival_date_week_number"].value_counts()).plot(figsize=(16,4))
plt.ylabel("Cancelations [%]")
plt.xlabel("Week number")
```

[407]: Text(0.5, 0, 'Week number')



- Per quanto riguarda mese e giorno della settimana sono presenti piccole fluttuazzioni
- Per quanto riguarda il numero della settimana dell'anno, possiamo notare periodi in cui il numero di prenotazioni cancellate rispetto al totale sono considerevoli

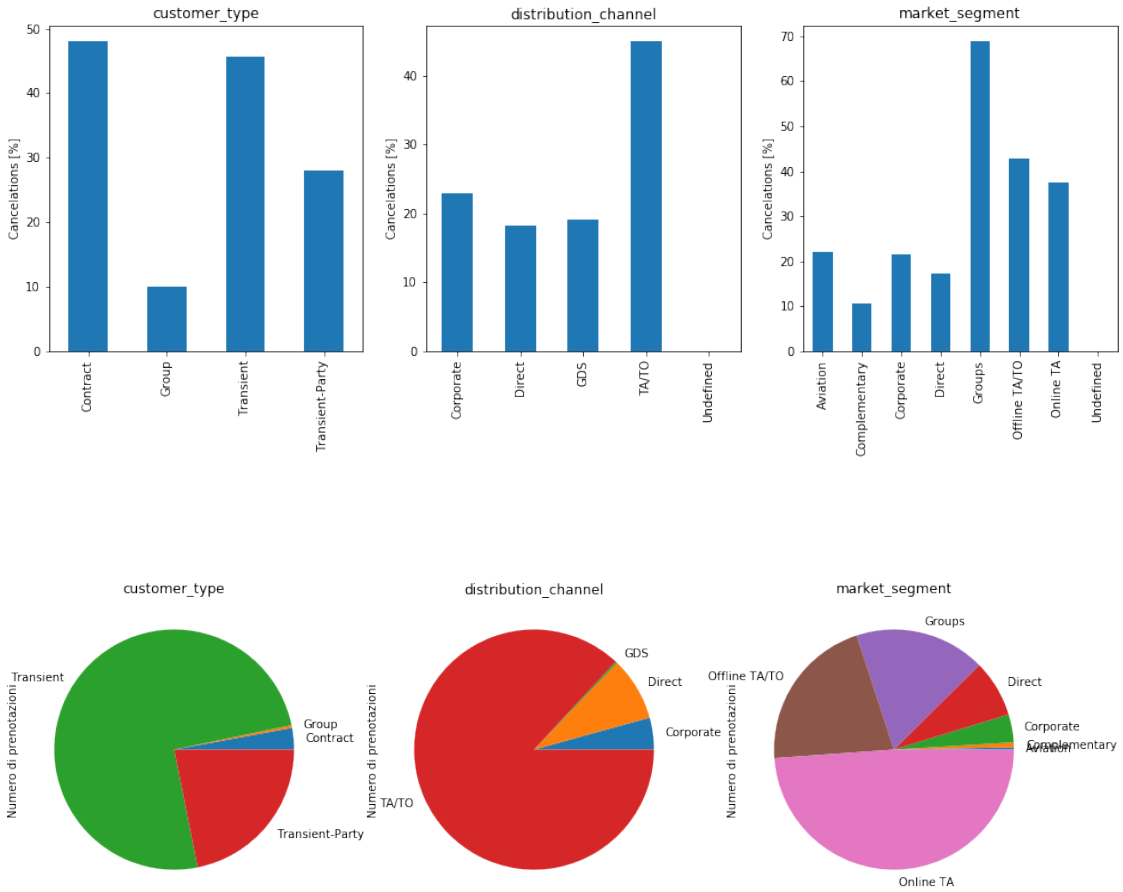
1.3.11 Qualche altra variabile ...

- Mostriamo infine rapidamente la percentuale di prenotazioni cancellate di qualche altra variabile, al fine di comprendere il dominio in cui lavoriamo correttamente
  - Assieme alle percentuali aggiungiamo anche la distribuzione delle prenotazioni

[408]:
```
fig = plt.figure(figsize=(16, 5))
ax1 = fig.add_subplot(131)
ax2 = fig.add_subplot(132)
ax3 = fig.add_subplot(133)
ax1.set(ylabel="Cancelations [%]")
ax2.set(ylabel="Cancelations [%]")
ax3.set(ylabel="Cancelations [%]")
ax1.set_title('customer_type')
ax2.set_title('distribution_channel')
ax3.set_title('market_segment')
(100 * hbd.groupby("customer_type").sum()["is_canceled"] / hbd["customer_type"].
 ↪value_counts()).plot.bar(ax=ax1)
(100 * hbd.groupby("distribution_channel").sum()["is_canceled"] /␣
 ↪hbd["distribution_channel"].value_counts()).plot.bar(ax=ax2)
(100 * hbd.groupby("market_segment").sum()["is_canceled"] /␣
 ↪hbd["market_segment"].value_counts()).plot.bar(ax=ax3)
fig = plt.figure(figsize=(16, 5))
ax4 = fig.add_subplot(131)
ax5 = fig.add_subplot(132)
ax6 = fig.add_subplot(133)
ax4.set_title('customer_type')
ax5.set_title('distribution_channel')
ax6.set_title('market_segment')
hbd.groupby("customer_type").size().plot.pie(ax=ax4)
```

```
hbd.groupby("distribution_channel").size().plot.pie(ax=ax5)
hbd.groupby("market_segment").size().plot.pie(ax=ax6)
ax4.set(ylabel="Numero di prenotazioni")
ax5.set(ylabel="Numero di prenotazioni")
ax6.set(ylabel="Numero di prenotazioni")
```

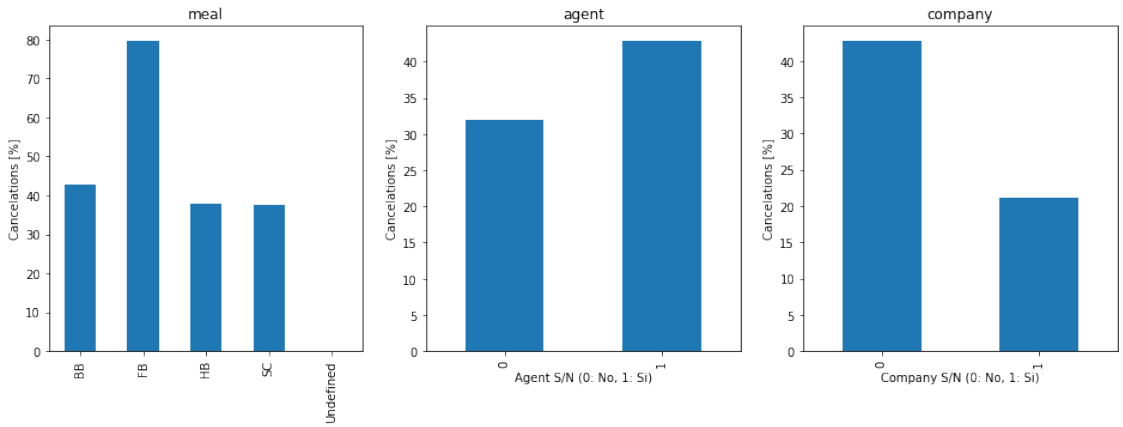
[408]:
```
[Text(0, 0.5, 'Numero di prenotazioni')]
```

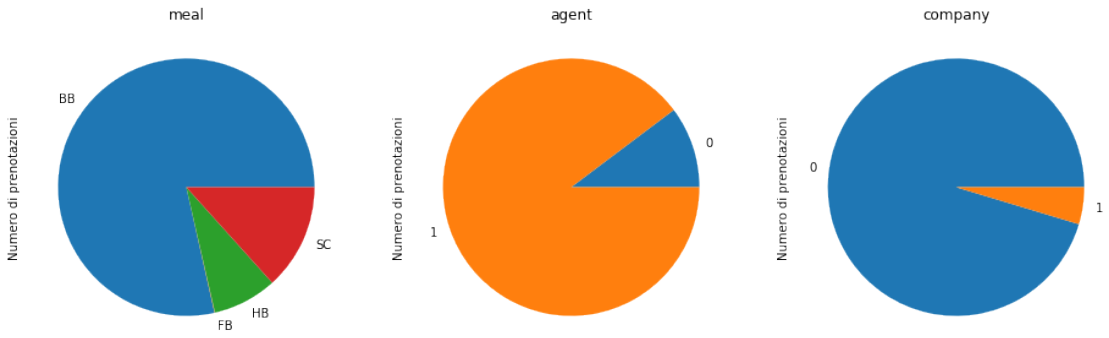


[409]:
```
fig = plt.figure(figsize=(16, 5))
ax4 = fig.add_subplot(131)
ax5 = fig.add_subplot(132)
ax6 = fig.add_subplot(133)
ax4.set_title('meal')
ax5.set_title('agent')
ax6.set_title('company')
ax4.set(ylabel="Cancelations [%]")
ax5.set(ylabel="Cancelations [%]")
ax6.set(ylabel="Cancelations [%]")
```

```
(100 * hbd.groupby("meal").sum()["is_canceled"] / hbd["meal"].value_counts()).
 ↪plot.bar(ax=ax4)
(100 * hbd.groupby("agent").sum()["is_canceled"] / hbd["agent"].value_counts()).
 ↪plot.bar(ax=ax5)
(100 * hbd.groupby("company").sum()["is_canceled"] / hbd["company"].
 ↪value_counts()).plot.bar(ax=ax6)
ax5.set(xlabel="Agent S/N (0: No, 1: Si)")
ax6.set(xlabel="Company S/N (0: No, 1: Si)")
fig = plt.figure(figsize=(16, 5))
ax4 = fig.add_subplot(131)
ax5 = fig.add_subplot(132)
ax6 = fig.add_subplot(133)
ax4.set_title('meal')
ax5.set_title('agent')
ax6.set_title('company')
hbd.groupby("meal").size().plot.pie(ax=ax4)
hbd.groupby("agent").size().plot.pie(ax=ax5)
hbd.groupby("company").size().plot.pie(ax=ax6)
ax4.set(ylabel="Numero di prenotazioni")
ax5.set(ylabel="Numero di prenotazioni")
ax6.set(ylabel="Numero di prenotazioni")
```

[409]: [Text(0, 0.5, 'Numero di prenotazioni')]





- Anche qui esistono valori con una percentuale di cancellazione molto elevata (fino all'80%) e altri con una percentuale molto bassa
- Per esempio…
  - Le prenotazioni con valore `FB` nella variabile `meal` sono state per circa l'80% cancellate

1.4 Classificazione

- Convertiamo, per maggiore chiarezza, i valori della variabile `is_canceled` (ovvero 0 e 1) in `N` e `Y`:
  - `0` diventa `N`, ovvero non cancellata
  - `1` diventa `Y`, ovvero cancellata

[410]:
```
hbd["is_canceled"] = hbd["is_canceled"].map(lambda value: "N" if value is 0␣
↪else "Y")
hbd["is_canceled"].unique()
```

[410]:
```
array(['N', 'Y'], dtype=object)
```

- Impostiamo come variabile da predire la classe `is_canceled` e come variabili predittive tutte le altre

[411]:
```
y = hbd["is_canceled"]
X = hbd.drop(columns="is_canceled")
```

- Molte variabili sono categoriche, è quindi necessario applicare la binarizzazione delle feature, ovvero convertire ciascuna di esse in una o più variabili binarie.
- La conversione viene eseguita in modo molto basilare dal comando `get_dummies`
  - NB: non sono previste variabili per tutti quei valori non presenti nel dataset

[412]:
```
X = pd.get_dummies(X)
```

- Otteniamo un numero di variabili pari a …

[413]:
```
X.shape[1]
```

[413]: 248

- Vengono mostrate per completezza tutte le variabili ottenute

[414]:
```
X.columns.tolist()
```

[414]:
```
['lead_time',
 'arrival_date_week_number',
 'stays_in_weekend_nights',
 'stays_in_week_nights',
 'adults',
 'children',
 'babies',
 'is_repeated_guest',
 'previous_cancellations',
 'previous_bookings_not_canceled',
 'booking_changes',
 'agent',
 'company',
 'days_in_waiting_list',
 'adr',
 'required_car_parking_spaces',
 'total_of_special_requests',
 'arrival_date_day_Friday',
 'arrival_date_day_Monday',
 'arrival_date_day_Saturday',
 'arrival_date_day_Sunday',
 'arrival_date_day_Thursday',
 'arrival_date_day_Tuesday',
 'arrival_date_day_Wednesday',
 'arrival_date_month_April',
 'arrival_date_month_August',
 'arrival_date_month_December',
 'arrival_date_month_February',
 'arrival_date_month_January',
 'arrival_date_month_July',
 'arrival_date_month_June',
 'arrival_date_month_March',
 'arrival_date_month_May',
 'arrival_date_month_November',
 'arrival_date_month_October',
 'arrival_date_month_September',
 'meal_BB',
 'meal_FB',
 'meal_HB',
 'meal_SC',
 'meal_Undefined',
```

```
'country_ABW',
'country_AGO',
'country_AIA',
'country_ALB',
'country_AND',
'country_ARE',
'country_ARG',
'country_ARM',
'country_ASM',
'country_ATA',
'country_ATF',
'country_AUS',
'country_AUT',
'country_AZE',
'country_BDI',
'country_BEL',
'country_BEN',
'country_BFA',
'country_BGD',
'country_BGR',
'country_BHR',
'country_BHS',
'country_BIH',
'country_BLR',
'country_BOL',
'country_BRA',
'country_BRB',
'country_BWA',
'country_CAF',
'country_CHE',
'country_CHL',
'country_CHN',
'country_CIV',
'country_CMR',
'country_CN',
'country_COL',
'country_COM',
'country_CPV',
'country_CRI',
'country_CUB',
'country_CYM',
'country_CYP',
'country_CZE',
'country_DEU',
'country_DJI',
'country_DMA',
'country_DNK',
```

```
'country_DOM',
'country_DZA',
'country_ECU',
'country_EGY',
'country_ESP',
'country_EST',
'country_ETH',
'country_FIN',
'country_FJI',
'country_FRA',
'country_FRO',
'country_GAB',
'country_GBR',
'country_GEO',
'country_GGY',
'country_GHA',
'country_GIB',
'country_GLP',
'country_GNB',
'country_GRC',
'country_GTM',
'country_GUY',
'country_HKG',
'country_HND',
'country_HRV',
'country_HUN',
'country_IDN',
'country_IMN',
'country_IND',
'country_IRL',
'country_IRN',
'country_IRQ',
'country_ISL',
'country_ISR',
'country_ITA',
'country_JAM',
'country_JEY',
'country_JOR',
'country_JPN',
'country_KAZ',
'country_KEN',
'country_KHM',
'country_KIR',
'country_KNA',
'country_KOR',
'country_KWT',
'country_LAO',
```

```
'country_LBN',
'country_LBY',
'country_LCA',
'country_LIE',
'country_LKA',
'country_LTU',
'country_LUX',
'country_LVA',
'country_MAC',
'country_MAR',
'country_MCO',
'country_MDG',
'country_MDV',
'country_MEX',
'country_MKD',
'country_MLI',
'country_MLT',
'country_MMR',
'country_MNE',
'country_MOZ',
'country_MRT',
'country_MUS',
'country_MWI',
'country_MYS',
'country_MYT',
'country_NAM',
'country_NCL',
'country_NGA',
'country_NIC',
'country_NLD',
'country_NOR',
'country_NPL',
'country_NZL',
'country_OMN',
'country_PAK',
'country_PAN',
'country_PER',
'country_PHL',
'country_PLW',
'country_POL',
'country_PRI',
'country_PRT',
'country_PRY',
'country_PYF',
'country_QAT',
'country_ROU',
'country_RUS',
```

```
'country_RWA',
'country_SAU',
'country_SDN',
'country_SEN',
'country_SGP',
'country_SLE',
'country_SLV',
'country_SMR',
'country_SRB',
'country_STP',
'country_SUR',
'country_SVK',
'country_SVN',
'country_SWE',
'country_SYC',
'country_SYR',
'country_TGO',
'country_THA',
'country_TJK',
'country_TMP',
'country_TUN',
'country_TUR',
'country_TWN',
'country_TZA',
'country_UGA',
'country_UKR',
'country_UMI',
'country_URY',
'country_USA',
'country_UZB',
'country_VEN',
'country_VGB',
'country_VNM',
'country_ZAF',
'country_ZMB',
'country_ZWE',
'market_segment_Aviation',
'market_segment_Complementary',
'market_segment_Corporate',
'market_segment_Direct',
'market_segment_Groups',
'market_segment_Offline TA/TO',
'market_segment_Online TA',
'market_segment_Undefined',
'distribution_channel_Corporate',
'distribution_channel_Direct',
'distribution_channel_GDS',
```

```
 'distribution_channel_TA/TO',
 'distribution_channel_Undefined',
 'reserved_room_type_A',
 'reserved_room_type_B',
 'reserved_room_type_C',
 'reserved_room_type_D',
 'reserved_room_type_E',
 'reserved_room_type_F',
 'reserved_room_type_G',
 'reserved_room_type_H',
 'reserved_room_type_L',
 'reserved_room_type_P',
 'deposit_type_No Deposit',
 'deposit_type_Non Refund',
 'deposit_type_Refundable',
 'customer_type_Contract',
 'customer_type_Group',
 'customer_type_Transient',
 'customer_type_Transient-Party']
```

- Suddividiamo i dati in un training set e in un validation set con la funzione `train_test_split` con proporzione 66-33

[415]:
```
from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(
    X, y,
    test_size=1/3,
    random_state=42
)
```

- Definiamo un modello di regressione logistica più semplice possibile, configurandone l'implementazione e il seed per la casualità
  - gli altri parametri sono lasciati ai valori di default, ad es. la regolarizzazione applicata è L2 con C=1

[416]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline

model = LogisticRegression(solver="saga", random_state=42)
```

- Addestriamo il modello sui dati

[417]:
```
%time model.fit(X_train, y_train)
```

```
Wall time: 21.3 s
```

```
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
```

```
        "the coef_ did not converge", ConvergenceWarning)
```

[417]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=42, solver='saga', tol=0.0001, verbose=0,
                   warm_start=False)
```

- Mostriamo le classi previste dal modello

[418]:
```
model.classes_
```

[418]:
```
array(['N', 'Y'], dtype=object)
```

- NB: Quando effettuiamo una predizione di probabilità otteniamo due valori ([a, b])
  - Il primo valore (a) si riferisce alla probabilità di ottenere la classe `N`
  - Il secondo valore (b) si riferisce alla probabilità di ottenere la classe `Y`
- Come mostrato di seguito...

[419]:
```
model.predict_proba(X_val[:3])
```

[419]:
```
array([[0.19018121, 0.80981879],
       [0.10774354, 0.89225646],
       [0.88131685, 0.11868315]])
```

[420]:
```
model.predict(X_val[:3])
```

[420]:
```
array(['Y', 'Y', 'N'], dtype=object)
```

- Definiamo una funzione per ottenere le informazioni utili per valutare un modello di classificazione
- Oltre all'accuratezza come percentuale di classificazioni corrette, esistono altri modi per valutare l'accuratezza di un classificatore
  - Precision e recall sono particolarmente utili in caso di sbilanciamento tra le classi, per cui l'accuratezza può non essere un indicatore affidabile

[421]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score, recall_score, f1_score

def print_model_informations(model,  X_train, y_train, X_val, y_val):
    y_pred = model.predict(X_val)
    print("Accuracy =", model.score(X_val, y_val),"        ( Accuracy on␣
 ↪training set =", model.score(X_train, y_train),")")
    print("\nPrecision (Y) =", precision_score(y_val,y_pred, pos_label="Y"))
    print("Precision (N) =", precision_score(y_val,y_pred, pos_label="N"))
    print("Precision =", precision_score(y_val,y_pred, average="macro"))
    print("\nRecall (Y) =", recall_score(y_val,y_pred, pos_label="Y"))
```

```
    print("Recall (N) =", recall_score(y_val,y_pred, pos_label="N"))
    print("Recall =", recall_score(y_val,y_pred, average="macro"))
    print("\nF1 Score (Y) =", f1_score(y_val,y_pred, pos_label="Y"))
    print("F1 Score (N) =", f1_score(y_val,y_pred, pos_label="N"))
    print("F1 Score =", f1_score(y_val,y_pred, average="macro"))
    print("\nMatrice di confusione:")
    cm = confusion_matrix(y_val, y_pred)
    print(pd.DataFrame(cm, index=model.classes_, columns=model.classes_))
```

- Calcoliamo le misure del nostro modello

[422]:
```
print_model_informations(model, X_train, y_train, X_val, y_val)
```

```
Accuracy = 0.7946205644246047          ( Accuracy on training set =
0.7960920801255982 )

Precision (Y) = 0.8481325748206777
Precision (N) = 0.7710377152823196
Precision = 0.8095851450514986

Recall (Y) = 0.6201284022063478
Recall (N) = 0.920130081300813
Recall = 0.7701292417535803

F1 Score (Y) = 0.7164272656045966
F1 Score (N) = 0.8390119503009816
F1 Score = 0.777719607952789

Matrice di confusione:
       N     Y
N  14147  1228
Y   4201  6858
```

- Per avere una valutazione più completa del modello ottenuto, possiamo metterlo a confronto con quello che accadrebbe prendendo decisioni casuali, ovvero dotandoci di un modello randomico
  - In questo caso *DummyClassifier*

[423]:
```
from sklearn.dummy import DummyClassifier
```

[424]:
```
random = DummyClassifier(strategy="uniform", random_state=42)
random.fit(X_train, y_train)
```

[424]:
```
DummyClassifier(constant=None, random_state=42, strategy='uniform')
```

[425]:
```
random.score(X_val, y_val)
```

[425]:
```
0.49708708481501096
```

- Abbiamo quindi ottenuto un modello che ci consente di intraprendere decisioni più accurate di come le faremmo casualmente

1.4.1 Standardizzazione

- Possiamo standardizzare i dati per vedere se il modello migliora

[426]:
```
from sklearn.preprocessing import StandardScaler

model_stand = Pipeline([
    ("scaler", StandardScaler()),
    ("log", LogisticRegression(solver="saga", random_state=42))
])
model_stand.fit(X_train, y_train)
print_model_informations(model_stand, X_train, y_train, X_val, y_val)
```

```
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
Accuracy = 0.8097904214269501         ( Accuracy on training set =
0.8101462159759396 )

Precision (Y) = 0.8362136247073252
Precision (N) = 0.7962210134554824
Precision = 0.8162173190814037

Recall (Y) = 0.6781806673297767
Recall (N) = 0.9044552845528455
Recall = 0.7913179759413111

F1 Score (Y) = 0.7489514679448771
F1 Score (N) = 0.8468940316686967
F1 Score = 0.7979227498067869

Matrice di confusione:
       N     Y
N  13906  1469
Y   3559  7500
```

- La standardizzazione comporta un miglioramento sotto più punti di vista

1.4.2 Regolarizzazione

- Nella regressione logistica possiamo applicare le teniche di regolarizzazione
- Verifichiamo utilizzando la regressione lasso se ci sono coefficienti che si azzerano per valori di alpha non elevati
  - E' spesso indice di collinearità

[427]:
```
model_reg_1 = Pipeline([
    ("scaler", StandardScaler()),
    ("log", LogisticRegression(solver="saga", random_state=42, penalty="l1",␣
 ↪C=1))
])
%time model_reg_1.fit(X_train, y_train)
```

```
Wall time: 27.8 s
```

```
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

[427]:
```
Pipeline(memory=None,
         steps=[('scaler',
                 StandardScaler(copy=True, with_mean=True, with_std=True)),
                ('log',
                 LogisticRegression(C=1, class_weight=None, dual=False,
                                    fit_intercept=True, intercept_scaling=1,
                                    l1_ratio=None, max_iter=100,
                                    multi_class='auto', n_jobs=None,
                                    penalty='l1', random_state=42,
                                    solver='saga', tol=0.0001, verbose=0,
                                    warm_start=False))],
         verbose=False)
```

[428]:
```
coeff = pd.Series(model_reg_1.named_steps["log"].coef_[0], index=X.columns)
', '.join(coeff[coeff==0].index)
```

[428]:
```
'meal_Undefined, country_BDI, country_BHS, country_BWA, country_CAF,
country_CYM, country_DJI, country_DMA, country_ESP, country_FJI, country_IMN,
country_KHM, country_KIR, country_LCA, country_MDG, country_MWI, country_NAM,
country_NPL, country_PLW, country_PYF, country_SDN, country_SMR, country_SUR,
country_TGO, country_VGB, country_ZMB, market_segment_Undefined,
distribution_channel_Undefined, reserved_room_type_H, reserved_room_type_L,
reserved_room_type_P'
```

- Sono state rimosse tutte le variabili *undefined*
  - Il valore undefined è un valore utilizzato quando non viene selezionato nessun valore per una certa variabile
    - Nel caso della variabile inziale `meal`, per esempio, viene assegnato se non seleziono nessuna combinazione di pasti
  - Di fatto questo tipo di variabile è deducibile dalle altre variabili ottenute, mediante binarizzazione, dalla stessa variabile categorica
- Per il resto nulla di particolarmente interessante, se non la rimozione di molte delle variabili binarie relative agli stati
  - Poichè sono presenti quasi tutti gli stati del mondo (~196) è un risultato comprensibile

- Applichiamo nuovamente la regolarizzazione lasso, ma questa volta utilizziamo un alpha più elevato al fine di annullare molte più variabili
  - Ci servirà per il prossimo step

[429]:
```
model_reg_2 = Pipeline([
    ("scaler", StandardScaler()),
    ("log", LogisticRegression(solver="saga", random_state=42, penalty="l1",␣
 ↪C=0.002))
])
%time model_reg_2.fit(X_train, y_train)
```

```
Wall time: 24.5 s
```

```
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

[429]:
```
Pipeline(memory=None,
         steps=[('scaler',
                 StandardScaler(copy=True, with_mean=True, with_std=True)),
                ('log',
                 LogisticRegression(C=0.002, class_weight=None, dual=False,
                                    fit_intercept=True, intercept_scaling=1,
                                    l1_ratio=None, max_iter=100,
                                    multi_class='auto', n_jobs=None,
                                    penalty='l1', random_state=42,
                                    solver='saga', tol=0.0001, verbose=0,
                                    warm_start=False))],
         verbose=False)
```

- Andiamo a visualizzare i coefficienti più grandi (in valore assoluto) per comprendere quali siano le feature più rilevanti

[430]:
```
pd.Series(model_reg_2.named_steps["log"].coef_[0], index=X.columns).
 ↪sort_values(ascending=False).head(10).append(
pd.Series(model_reg_2.named_steps["log"].coef_[0], index=X.columns).
 ↪sort_values(ascending=False).tail(10))
```

[430]:
```
deposit_type_Non Refund           0.644021
market_segment_Online TA          0.512518
country_PRT                       0.511609
lead_time                         0.436877
previous_cancellations            0.227620
adr                               0.135569
customer_type_Transient           0.116751
stays_in_week_nights              0.110001
distribution_channel_TA/TO        0.051753
```

```
country_CHN                          0.051374
company                             -0.038286
market_segment_Offline TA/TO        -0.049784
customer_type_Transient-Party       -0.062018
previous_bookings_not_canceled      -0.079668
country_FRA                         -0.097700
country_DEU                         -0.149741
booking_changes                     -0.159968
required_car_parking_spaces         -0.287879
total_of_special_requests           -0.517619
deposit_type_No Deposit             -0.645471
dtype: float64
```

- Visualizziamo anche l'intercetta

[431]:
```
model_reg_2.named_steps["log"].intercept_
```

[431]:
```
array([-0.18286659])
```

- Verifichiamo quanti coefficienti hanno valore diverso da zero

[432]:
```
coeff = pd.Series(model_reg_2.named_steps["log"].coef_[0], index=X.columns)
coeff_not_zero = coeff[coeff!=0]
print(len(coeff_not_zero),"coefficienti hanno valore diverso da 0")
```

```
35 coefficienti hanno valore diverso da 0
```

- Le variabili con coefficiente più alto (in valore assoluto) mostrate sopra sono state soggetto di analisi in fase esplorativa in quanto considerate le più rilevanti
  - E' stato quindi tenuto in considerazione anche questo risultato per decidere quali variabili analizzare in fase esplorativa

Variabile `deposit_type`

- Poichè `deposit_type` risulta essere una variabile di rilievo per il modello (e tenuto conto delle osservazioni fatte in fase di analisi esplorativa) si trova opportuno sperimentare l'addestramento di un modello senza essa
  - Effettuiamo una copia dei dataset `X_train` e `X_val` e rimuoviamo la variabile `deposit_type`

[433]:
```
X_train_no_deposit = X_train.copy()
X_val_no_deposit = X_val.copy()
X_train_no_deposit.drop(columns=['deposit_type_No Deposit', 'deposit_type_Non␣
↪Refund','deposit_type_Refundable'], inplace=True)
X_val_no_deposit.drop(columns=['deposit_type_No Deposit', 'deposit_type_Non␣
↪Refund','deposit_type_Refundable'], inplace=True)
```

- Creiamo il nuovo modello con la stessa configurazione di `model_stand` e lo addestriamo con una variabile in meno

[434]:
```
model_no_deposit = Pipeline([
    ("scaler", StandardScaler()),
    ("log", LogisticRegression(solver="saga", random_state=42))
])
model_no_deposit.fit(X_train_no_deposit, y_train)
```

```
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

[434]:
```
Pipeline(memory=None,
         steps=[('scaler',
                 StandardScaler(copy=True, with_mean=True, with_std=True)),
                ('log',
                 LogisticRegression(C=1.0, class_weight=None, dual=False,
                                    fit_intercept=True, intercept_scaling=1,
                                    l1_ratio=None, max_iter=100,
                                    multi_class='auto', n_jobs=None,
                                    penalty='l2', random_state=42,
                                    solver='saga', tol=0.0001, verbose=0,
                                    warm_start=False))],
         verbose=False)
```

- Visualizziamo le metriche

[435]:
```
print_model_informations(model_no_deposit, X_train_no_deposit, y_train,␣
↪X_val_no_deposit, y_val)
```

```
Accuracy = 0.8001815843232201         ( Accuracy on training set =
0.7994968505873229 )

Precision (Y) = 0.7922695537792168
Precision (N) = 0.8049060479729322
Precision = 0.7985878008760745

Recall (Y) = 0.7080206166922868
Recall (N) = 0.8664715447154472
Recall = 0.787246080703867

F1 Score (Y) = 0.7477795817018431
F1 Score (N) = 0.834554908225271
F1 Score = 0.7911672449635571

Matrice di confusione:
       N     Y
N  13322  2053
Y   3229  7830
```

- Il modello non sembra aver subito in modo rilevante la mancanza della variabile `deposit_type`

1.4.3 Cross-validation

- Quello che vogliamo fare ora è applicare la **Grid Search** e **K-fold cross validation** per trovare gli iperparametri migliori
- Poichè il dataset è molto ampio e le variabili sono molte, la ricerca degli iperparametri ottimali risulta essere molto dispendiosa.
- Riduciamo la dimensione del dataset in termini di variabili
- Realizzo una copia dei dataset di training e validation

[436]:
```
X_train_v2 = X_train.copy()
y_train_v2 = y_train.copy()
X_val_v2 = X_val.copy()
y_val_v2 = y_val.copy()
```

- Attualmente presentano le seguenti dimensioni...

[437]:
```
X_train_v2.shape, X_val_v2.shape
```

[437]:
```
((52867, 248), (26434, 248))
```

- Utilizziamo la serie `coeff_not_zero` ottenuta precedentemente
  - Contiene i coefficienti diversi da zero ottenuti dall'addestramento di un modello di regressione logistica con regolarizzazione Lasso, alpha = 0.003 e variabili standardizzate
- Rimuoviamo dalle copie dei dataset tutte le variabili non presenti all'interno della serie `coeff_not_zero`
  - Vengono quindi rimosse dai nostri dataset le variabili meno rilevanti per il modello precedente

[438]:
```
X_train_v2 = X_train_v2[coeff_not_zero.index]
X_val_v2 = X_val_v2[coeff_not_zero.index]
print("Ho scartato", X_train.shape[1] - coeff_not_zero.shape[0], "variabili")
```

```
Ho scartato 213 variabili
```

- Le nuove dimensioni sono:

[439]:
```
X_train_v2.shape, X_val_v2.shape
```

[439]:
```
((52867, 35), (26434, 35))
```

- Addestriamo un modello su questo nuovo training set
  - Viene utilizzata la configurazione di `model_stand`

[440]:
```
model_stand_v2 = Pipeline([
    ("scaler", StandardScaler()),
    ("log", LogisticRegression(solver="saga", random_state=42))
```

```
])
model_stand_v2.fit(X_train_v2, y_train_v2)
```

```
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

[440]:
```
Pipeline(memory=None,
         steps=[('scaler',
                 StandardScaler(copy=True, with_mean=True, with_std=True)),
                ('log',
                 LogisticRegression(C=1.0, class_weight=None, dual=False,
                                    fit_intercept=True, intercept_scaling=1,
                                    l1_ratio=None, max_iter=100,
                                    multi_class='auto', n_jobs=None,
                                    penalty='l2', random_state=42,
                                    solver='saga', tol=0.0001, verbose=0,
                                    warm_start=False))],
         verbose=False)
```

- Visulizziamone le metriche...

[441]:
```
print_model_informations(model_stand_v2, X_train_v2, y_train_v2, X_val_v2,␣
↪y_val_v2)
```

```
Accuracy = 0.8071801467806613         ( Accuracy on training set =
0.8045472601055479 )

Precision (Y) = 0.8381352087114338
Precision (N) = 0.7916903167215348
Precision = 0.8149127627164843

Recall (Y) = 0.668143593453296
Recall (N) = 0.9071869918699187
Recall = 0.7876652926616073

F1 Score (Y) = 0.7435471698113209
F1 Score (N) = 0.8455126845088352
F1 Score = 0.7945299271600781

Matrice di confusione:
       N     Y
N  13948  1427
Y   3670  7389
```

- E confrontiamole con quelle ottenute dalla stessa configurazione addestrata però sul dataset completo di tutte le variabili

[442]:
```
print_model_informations(model_stand, X_train, y_train, X_val, y_val)
```

```
Accuracy = 0.8097904214269501         ( Accuracy on training set =
0.8101462159759396 )

Precision (Y) = 0.8362136247073252
Precision (N) = 0.7962210134554824
Precision = 0.8162173190814037

Recall (Y) = 0.6781806673297767
Recall (N) = 0.9044552845528455
Recall = 0.7913179759413111

F1 Score (Y) = 0.7489514679448771
F1 Score (N) = 0.8468940316686967
F1 Score = 0.7979227498067869

Matrice di confusione:
       N     Y
N  13906  1469
Y   3559  7500
```

- Il modello non sembra aver subito la mancanza delle variabili, le metriche sono estremamente simili
- Definiamo uno `StatifiedKFold` per effettuare la cross-validation
  - Dovendo addestrare un modello a riconoscere delle classi, è opportuno che le proporzioni di ciascuna classe nei fold siano uguali
  - `StratifiedKFold` è una variante di `KFold` che garantisce uguale distribuzione delle classi tra un fold e l'altro

[443]:
```
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.metrics import f1_score
skf = StratifiedKFold(3, shuffle=True, random_state=42)
```

[444]:
```
for train, val in skf.split(X_train_v2, y_train_v2):
    print(y_train.iloc[val].value_counts())
```

```
N    10284
Y     7339
Name: is_canceled, dtype: int64
N    10284
Y     7338
Name: is_canceled, dtype: int64
N    10283
Y     7339
Name: is_canceled, dtype: int64
```

- Definiamo una "griglia" con liste di valori possibili per gli iperparametri di un modello, al fine di testare tutte le combinazioni possibili mediante la grid search
  - Il numero di istanze combinato al numero di variabili rende impraticabile l'uso di feature polinomiali all'interno della Grid Search

[445]:
```
mod = Pipeline([
    ("scaler", None),
    ("lr", LogisticRegression(solver="saga", random_state=42))
])
grid = [
    {
        "scaler": [None, StandardScaler()],
        "lr__penalty": ["none"]
    },
    {
        "scaler": [None, StandardScaler()],
        "lr__penalty": ["l2", "l1"],
        "lr__C": np.logspace(-2, 2, 5)
    },
    {
        "scaler": [None, StandardScaler()],
        "lr__penalty": ["elasticnet"],
        "lr__C": np.logspace(-2, 2, 5),
        "lr__l1_ratio": [0.2, 0.5, 0.7]
    }
]
```

- Definiamo la grid search, specificando il modello, la lista di griglie e lo splitter per la cross-validation (usiamo lo `StratifiedKFold` creato sopra)

[446]:
```
gs = GridSearchCV(mod, grid, cv=skf)
```

- Effettuiamo quindi la ricerca sui dati

[447]:
```
%time gs.fit(X_train_v2, y_train_v2)
```

```
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

```
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

```
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

```
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

```
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

```
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

```
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

```
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

```
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

```
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

```
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

```
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

```
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

```
Wall time: 7min 52s
```

```
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

[447]:
```
GridSearchCV(cv=StratifiedKFold(n_splits=3, random_state=42, shuffle=True),
             error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('scaler', None),
```

```
                                        ('lr',
                                         LogisticRegression(C=1.0,
                                                            class_weight=None,
                                                            dual=False,
                                                            fit_intercept=True,
                                                            intercept_scaling=1,
                                                            l1_ratio=None,
                                                            max_iter=100,
                                                            multi_class='auto',
                                                            n_jobs=None,
                                                            penalty='l2',
                                                            random_state=42,
                                                            solver='saga',
                                                            tol=0.0…
                                'lr__penalty': ['l2', 'l1'],
                                'scaler': [None,
                                           StandardScaler(copy=True, with_mean=True,
                                                          with_std=True)]},
                               {'lr__C': array([1.e-02, 1.e-01, 1.e+00, 1.e+01,
       1.e+02]),
                                'lr__l1_ratio': [0.2, 0.5, 0.7],
                                'lr__penalty': ['elasticnet'],
                                'scaler': [None,
                                           StandardScaler(copy=True, with_mean=True,
                                                          with_std=True)]}],
                    pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                    scoring=None, verbose=0)
```

- Possiamo verificare la migliore combinazione di iperparametri
  - La metrica di riferimento di default è l'accuratezza, cioè la percentuale di classificazioni corrette

[448]:
```
gs.best_params_
```

[448]:
```
{'lr__C': 1.0,
 'lr__penalty': 'l1',
 'scaler': StandardScaler(copy=True, with_mean=True, with_std=True)}
```

- E vedere tutti i dettagli
  - Selezioniamo le 5 parametrizzazioni con accuratezza migliore

[449]:
```
result = pd.DataFrame(gs.cv_results_)
result.sort_values(by="rank_test_score", inplace=True)
result.reset_index(inplace=True, drop=True)
result.head(5)
```

[449]:
```
   mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
0       2.897346      0.009732         0.043995        0.000816
```

```
1       3.400340      0.037106         0.044994        0.000008
2       3.417679      0.038326         0.044654        0.001249
3       3.542333      0.027196         0.044667        0.000942
4       3.452005      0.026096         0.042993        0.000808

  param_lr__penalty                                      param_scaler  \
0                l1  StandardScaler(copy=True, with_mean=True, with…
1        elasticnet  StandardScaler(copy=True, with_mean=True, with…
2        elasticnet  StandardScaler(copy=True, with_mean=True, with…
3        elasticnet  StandardScaler(copy=True, with_mean=True, with…
4        elasticnet  StandardScaler(copy=True, with_mean=True, with…

  param_lr__C param_lr__l1_ratio  \
0           1                NaN
1           1                0.7
2           1                0.5
3         0.1                0.2
4           1                0.2

                                              params  split0_test_score  \
0  {'lr__C': 1.0, 'lr__penalty': 'l1', 'scaler': …           0.806049
1  {'lr__C': 1.0, 'lr__l1_ratio': 0.7, 'lr__penal…           0.806049
2  {'lr__C': 1.0, 'lr__l1_ratio': 0.5, 'lr__penal…           0.805992
3  {'lr__C': 0.1, 'lr__l1_ratio': 0.2, 'lr__penal…           0.806162
4  {'lr__C': 1.0, 'lr__l1_ratio': 0.2, 'lr__penal…           0.805935

   split1_test_score  split2_test_score  mean_test_score  std_test_score  \
0           0.804279           0.804733         0.805020        0.000751
1           0.804279           0.804733         0.805020        0.000751
2           0.804279           0.804733         0.805001        0.000725
3           0.804506           0.804335         0.805001        0.000824
4           0.804279           0.804733         0.804982        0.000699

   rank_test_score
0                1
1                1
2                3
3                4
4                5
```

- Andiamo a visualizzare le misure del miglior modello ottenuto

[450]:
```
print_model_informations(gs, X_train_v2, y_train_v2, X_val_v2, y_val_v2)
```

```
Accuracy = 0.8069531663766362         ( Accuracy on training set =
0.8044148523653697 )

Precision (Y) = 0.837948252383114
```

```
Precision (N) = 0.7914538644875724
Precision = 0.8147010584353431

Recall (Y) = 0.6676914730084095
Recall (N) = 0.9071219512195122
Recall = 0.7874067121139608

F1 Score (Y) = 0.7431935987116906
F1 Score (N) = 0.8453495772342942
F1 Score = 0.7942715879729924

Matrice di confusione:
       N     Y
N  13947  1428
Y   3675  7384
```

- Possiamo notare un valore leggermente più basso degli altri per quanto riguarda `Recall (Y)`, ovvero la percentuale di istanze cancellate che sono state classificate come tali.
  - Dal suo valore sappiamo che di tutte le prenotazioni cancellate il modello è in grado di trovarne il 67% circa
  - In compenso `Precision (Y)` è alta, sappiamo quindi che circa l'84% delle istanze che vengono classificate come cancellate lo sono veramente

1.4.4 Test polinomiale con riduzione delle feature

- Proviamo come ultima strada a vedere se è possibile ottenere un modello migliore utilizzando le feature polinomiali
  - Come detto precedentemente il numero di istanze combinato al numero di variabili rende impraticabile l'uso di feature polinomiali
  - Riduciamo quindi ulteriormente il numero di feature
    - Per farlo utilizziamo il medesimo approccio proposto precedentemente: utilizziamo la regressione Lasso con un valore molto alto per rimuovere più variabili possibili, mantenendo le più rilevanti

[451]:
```
model_reg_extreme = Pipeline([
    ("scaler", StandardScaler()),
    ("log", LogisticRegression(solver="saga", random_state=42, penalty="l1",␣
 ↪C=0.0005))
])
%time model_reg_extreme.fit(X_train, y_train)
```

```
Wall time: 16.5 s
```

[451]:
```
Pipeline(memory=None,
         steps=[('scaler',
                 StandardScaler(copy=True, with_mean=True, with_std=True)),
                ('log',
                 LogisticRegression(C=0.0005, class_weight=None, dual=False,
```

```
                                       fit_intercept=True, intercept_scaling=1,
                                       l1_ratio=None, max_iter=100,
                                       multi_class='auto', n_jobs=None,
                                       penalty='l1', random_state=42,
                                       solver='saga', tol=0.0001, verbose=0,
                                       warm_start=False))],
         verbose=False)
```

- In numero di coefficienti diversi da 0 è...

[452]:
```
coeff = pd.Series(model_reg_extreme.named_steps["log"].coef_[0], index=X.
↪columns)
coeff_not_zero = coeff[coeff!=0]
len(coeff_not_zero)
```

[452]:
```
11
```

- Viene realizzata una copia del dataset binarizzato iniziale, rimuovendo tutte le variabili non presenti fra quelle con coefficiente diverso da 0

[453]:
```
X_train_poly = X_train[coeff_not_zero.index]
X_val_poly = X_val[coeff_not_zero.index]
```

- Si propone una Grid Search con features polinomiali, giusto per provare un paio di gradi possibili

[454]:
```
from sklearn.preprocessing import PolynomialFeatures
polynomial_model = Pipeline([
    ("poly", PolynomialFeatures(include_bias=False)),
    ("scale",  StandardScaler()),
    ("linreg", LogisticRegression(solver="saga", random_state=42))
])
grid = {
    "poly__degree": [2,3]
    }
gs_poly = GridSearchCV(polynomial_model, param_grid=grid)
%time gs_poly.fit(X_train_poly, y_train)
```

```
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
```

```
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

```
Wall time: 3min 6s
```

```
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

[454]:
```
GridSearchCV(cv=None, error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('poly',
                                        PolynomialFeatures(degree=2,
                                                           include_bias=False,
interaction_only=False,
                                                           order='C')),
                                       ('scale',
                                        StandardScaler(copy=True,
                                                       with_mean=True,
```

```
                                                with_std=True)),
                                       ('linreg',
                                        LogisticRegression(C=1.0,
                                                           class_weight=None,
                                                           dual=False,
                                                           fit_intercept=True,
                                                           intercept_scaling=1,
                                                           l1_ratio=None,
                                                           max_iter=100,
                                                           multi_class='auto',
                                                           n_jobs=None,
                                                           penalty='l2',
                                                           random_state=42,
                                                           solver='saga',
                                                           tol=0.0001,
                                                           verbose=0,
                                                           warm_start=False))],
                                verbose=False),
             iid='deprecated', n_jobs=None, param_grid={'poly__degree': [2, 3]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

- Mostriamo tutte le parametrizzazioni ottenute

[455]:
```
result_poly = pd.DataFrame(gs_poly.cv_results_)
result_poly.sort_values(by="rank_test_score", inplace=True)
result_poly.reset_index(inplace=True, drop=True)
result_poly
```

[455]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_poly__degree	params	split0_test_score	split1_test_score	split2_test_score	split3_test_score	split4_test_score	mean_test_score	std_test_score	rank_test_score
0	24.253602	0.050159	0.122797	0.000751	3	{'poly__degree': 3}	0.804899	0.805845	0.801192	0.806583	0.805164	0.804736	0.001866	1
1	6.198207	0.041719	0.042393	0.000809	2	{'poly__degree': 2}	0.804899	0.806601	0.800719	0.805259	0.806015	0.804699	0.002076	2

- Prendiamo il miglior modello e visualizziamone le metriche

[456]:
```
print_model_informations(gs_poly, X_train_poly, y_train, X_val_poly, y_val)
```

```
Accuracy = 0.8052508133464478          ( Accuracy on training set =
0.8056821835927894 )

Precision (Y) = 0.8265385040327036
Precision (N) = 0.7941667146062245
Precision = 0.810352609319464

Recall (Y) = 0.6764626096392079
Recall (N) = 0.8978861788617886
Recall = 0.7871743942504983

F1 Score (Y) = 0.7440079562406763
F1 Score (N) = 0.8428475486903961
F1 Score = 0.7934277524655362

Matrice di confusione:
       N     Y
N  13805  1570
Y   3578  7481
```

- Le feature polinomiali non hanno portato a grossi benefici

1.5 Valutazione dei modelli di classificazione

1.5.1 Intervallo di confidenza sui modelli

[457]:
```
from scipy.stats import norm
```

- Definiamo una funzione `conf_interval` che calcoli gli estremi dell'intervallo di confidenza e restituisca una tupla con i due estremi, dove:
  - a è l'accuratezza del modello misurata sul validation set
  - N è il numero di osservazioni nel validation set
  - Z è il valore tale per cui l'area sottesa dalla densità di probabilità $\varphi(x)$ della distribuzione normale standard tra -Z e Z sia il livello di confidenza 1-
- Poichè a noi interessa valutare i modelli con una condifidenza del 95%, possiamo ricavare dalle apposite tabelle di valori che, per 1- $= 0.95$ ($=0.05$), Z $= 1.96$

[458]:
```
def conf_interval(a, N, Z=1.96):
    c = (2 * N * a + Z**2) / (2 * (N + Z**2))
    d = Z * np.sqrt(Z**2 + 4*N*a - 4*N*a**2) / (2 * (N + Z**2))
    return c - d, c + d
```

- Definisco ora una funzione `model_conf_interval` in modo che:
  - prenda in input un modello addestrato `model`, un validation set `X, y` e un livello di confidenza `level` (default 0.95)
  - restituisca l'intervallo di confidenza dell'accuratezza del modello, servendosi della funzione `conf_interval` sopra

[459]:
```
def model_conf_interval(model, X, y, level=0.95):
    a = model.score(X, y)
    N = X.shape[0]
    Z = norm.ppf((1 + level) / 2)
    return conf_interval(a, N, Z)
```

1.5.2 Confronto tra modelli

- Dati due modelli diversi, vogliamo poter valutare se l'accuratezza 1 misurata su uno sia significativamente migliore della 2 misurata sull'altro.
- Implementiamo la funzione `diff_interval` in modo che
  - prenda in input le accuratezze `a1` e `a2`, i numeri di osservazioni `N1` e `N2` e il coefficiente `Z`
  - calcoli l'intervallo di confidenza della differenza tra due modelli secondo la formula sopra

[460]:
```
def diff_interval(a1, a2, N1, N2, Z):
    d = abs(a1 - a2)
    sd = np.sqrt(a1 * (1-a1) / N1 + a2 * (1-a2) / N2)
    return d - Z * sd, d + Z * sd
```

- Implementiamo la funzione `model_diff_interval` in modo che
  - prenda in input due modelli `m1, m2`, un validation set `X, y` e un livello di confidenza `level` (default 0.95)
  - restituisca l'intervallo di confidenza della differenza di accuratezza tra i due modelli, valutati entrambi sul validation set dato

[461]:
```
def model_diff_interval(m1, m2, X, y, level=0.95):
    a1 = m1.score(X, y)
    a2 = m2.score(X, y)
    N = len(X)
    Z = norm.ppf((1 + level) / 2)
    return diff_interval(a1, a2, N, N, Z)
```

1.5.3 Miglior modello della Grid Search

- Andiamo a prendere i parametri dei tre modelli testati che hanno ottenuto i valori *rank_test_score* più alti nella prima Grid Search
- Primo in classifica

[462]:
```
result.loc[0, 'params']
```

[462]:
```
{'lr__C': 1.0,
 'lr__penalty': 'l1',
 'scaler': StandardScaler(copy=True, with_mean=True, with_std=True)}
```

- Secondo in classifica

[463]:
```
result.loc[1, 'params']
```

[463]:
```
{'lr__C': 1.0,
 'lr__l1_ratio': 0.7,
 'lr__penalty': 'elasticnet',
 'scaler': StandardScaler(copy=True, with_mean=True, with_std=True)}
```

- Terzo in classifica

[464]:
```
result.loc[2, 'params']
```

[464]:
```
{'lr__C': 1.0,
 'lr__l1_ratio': 0.5,
 'lr__penalty': 'elasticnet',
 'scaler': StandardScaler(copy=True, with_mean=True, with_std=True)}
```

- Dai parametri mostrati genero tre modelli, i tre modelli ipoteticamente migliori

[465]:
```
import copy
model_1 = copy.deepcopy(gs.best_estimator_.set_params(**result.loc[0,␣
↪'params']))
model_2 = copy.deepcopy(gs.best_estimator_.set_params(**result.loc[1,␣
↪'params']))
model_3 = copy.deepcopy(gs.best_estimator_.set_params(**result.loc[2,␣
↪'params']))
```

- Addestriamo i tre modelli sul training set

[466]:
```
%time model_1.fit(X_train_v2, y_train_v2)
```

```
Wall time: 5.21 s

C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

[466]:
```
Pipeline(memory=None,
         steps=[('scaler',
                 StandardScaler(copy=True, with_mean=True, with_std=True)),
                ('lr',
                 LogisticRegression(C=1.0, class_weight=None, dual=False,
                                    fit_intercept=True, intercept_scaling=1,
                                    l1_ratio=None, max_iter=100,
                                    multi_class='auto', n_jobs=None,
                                    penalty='l1', random_state=42,
                                    solver='saga', tol=0.0001, verbose=0,
                                    warm_start=False))],
         verbose=False)
```

[467]:
```
%time model_2.fit(X_train_v2, y_train_v2)
```

```
Wall time: 5.78 s

C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

[467]:
```
Pipeline(memory=None,
         steps=[('scaler',
                 StandardScaler(copy=True, with_mean=True, with_std=True)),
                ('lr',
                 LogisticRegression(C=1.0, class_weight=None, dual=False,
                                    fit_intercept=True, intercept_scaling=1,
                                    l1_ratio=0.7, max_iter=100,
                                    multi_class='auto', n_jobs=None,
                                    penalty='elasticnet', random_state=42,
                                    solver='saga', tol=0.0001, verbose=0,
                                    warm_start=False))],
         verbose=False)
```

[468]:
```
%time model_3.fit(X_train_v2, y_train_v2)
```

```
Wall time: 5.93 s

C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

[468]:
```
Pipeline(memory=None,
         steps=[('scaler',
                 StandardScaler(copy=True, with_mean=True, with_std=True)),
                ('lr',
                 LogisticRegression(C=1.0, class_weight=None, dual=False,
                                    fit_intercept=True, intercept_scaling=1,
                                    l1_ratio=0.5, max_iter=100,
                                    multi_class='auto', n_jobs=None,
                                    penalty='elasticnet', random_state=42,
                                    solver='saga', tol=0.0001, verbose=0,
                                    warm_start=False))],
         verbose=False)
```

- Usiamo la funzione `model_conf_interval` per calcolare l'intervallo di confidenza al 95% dell'accuratezza dei tre modelli ottenuti stimata sul validation set

[469]:
```
model_conf_interval(model_1, X_val_v2, y_val_v2)
```

[469]:
```
(0.8021507272563478, 0.8116664039673401)
```

[470]:
```
model_conf_interval(model_2, X_val_v2, y_val_v2)
```

[470]:
```
(0.802227085685336, 0.8117413438206353)
```

[471]:
```
model_conf_interval(model_3, X_val_v2, y_val_v2)
```

[471]:
```
(0.802227085685336, 0.8117413438206353)
```

- Utilizziamo `model_diff_interval` per calcolare l'intervallo in cui si colloca la differenza di accuratezza dei tre modelli ottenuti, calcolata sul validation set, al 95% di confidenza

[472]:
```
model_diff_interval(model_1, model_2, X_val_v2, y_val_v2)
```

[472]:
```
(-0.006652630965746836, 0.006803951235096778)
```

[473]:
```
model_diff_interval(model_2, model_3, X_val_v2, y_val_v2)
```

[473]:
```
(-0.006727789408089268, 0.006727789408089268)
```

[474]:
```
model_diff_interval(model_1, model_3, X_val_v2, y_val_v2)
```

[474]:
```
(-0.006652630965746836, 0.006803951235096778)
```

- In tutti e tre i casi non abbiamo la certezza che un modello sia meglio dell'altro
  - Poichè l'intervallo ottenuto include lo zero (l'estremo inferiore è negativo), non abbiamo la certezza al 95% che il modello con accuratezza stimata maggiore sia effettivamente migliore
  - Alle volte può essere meglio uno alle vole l'altro
- Prendiamo quindi come riferimento il primo, in quanto tale

Addestramento miglior configurazione della Grid Search su tutte le variabili

- Come parte di questo studio possiamo prendere la migliore configurazione ottenuta dalla grid search ed addestrarla su tutte le variabili invece che solo su una parte di esse

[475]:
```
model_1_completo = copy.deepcopy(gs.best_estimator_.set_params(**result.loc[0,␣
↪'params']))
```

- Addestriamo il modello su tutte le variabili

[476]:
```
%time model_1_completo.fit(X_train, y_train)
```

```
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_logistic.py:1501: UserWarning: l1_ratio parameter
is only used when penalty is 'elasticnet'. Got (penalty=l1)
  "(penalty={})".format(self.penalty))

Wall time: 27.6 s
```

```
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

[476]:
```
Pipeline(memory=None,
         steps=[('scaler',
                 StandardScaler(copy=True, with_mean=True, with_std=True)),
                ('lr',
                 LogisticRegression(C=1.0, class_weight=None, dual=False,
                                    fit_intercept=True, intercept_scaling=1,
                                    l1_ratio=0.5, max_iter=100,
                                    multi_class='auto', n_jobs=None,
                                    penalty='l1', random_state=42,
                                    solver='saga', tol=0.0001, verbose=0,
                                    warm_start=False))],
         verbose=False)
```

- Vogliamo confrontare il modello ottenuto con la sua stessa configurazione addestrata solo su parte delle variabili
  - Per fare questo è necessario definire una nuova funzione che chiamiamo `model_diff_interval_v2`
    - Implementiamo la funzione `model_diff_interval_v2` in modo che prenda in input due validation set, `Xm1, ym1` e `Xm2, ym2`, uno per ciascun modello.
    - Questo è necessario perchè i due modelli sono stati addestrati due dataset di training diversi, uno con più variabili ed uno con meno
      - Il confronto è valido se i due validation set passati sono uguali, ciò che può cambiare è solo il numero di variabili.

[477]:
```
def model_diff_interval_v2(m1, m2, Xm1, ym1, Xm2, ym2, level=0.95):
    a1 = m1.score(Xm1, ym1)
    a2 = m2.score(Xm2, ym2)
    N = len(Xm1)
    Z = norm.ppf((1 + level) / 2)
    return diff_interval(a1, a2, N, N, Z)
```

- Per ciascuna differenza vogliamo visualizzare:
  - le accurattezze di entrambi i modelli
  - se la differenza di accuratezza è statisticamente significativa
- Definiamo la funzione `valuate_complete_model` a cui passiamo:
  - Il primo modello e il suo validation set: `X_val_1`, `y_val_1`, `model_1`
  - Il secondo modello e il suo validation set: `X_val_2`, `y_val_2`, `model_2`

[478]:
```
def valuate_complete_model(model_1, X_val_1, y_val_1, model_2, X_val_2,␣
↪y_val_2):
    print("Accuratezza del primo modello: ", model_1.score(X_val_1, y_val_1))
    print("Accuratezza del secondo modello: ", model_2.score(X_val_2, y_val_2))
```

```
    print("Intervallo di confidenza: ", model_diff_interval_v2(model_1,␣
↪model_2, X_val_1, y_val_1, X_val_2, y_val_2))
```

- Verichiamo la differenza del modello appena addestrato su tutte le variabili con la sua versione addestrata solo su parte delle variabili

[479]:
```
valuate_complete_model(model_1_completo, X_val, y_val, model_1, X_val_v2,␣
↪y_val_v2)
```

```
Accuratezza del primo modello:  0.8100552318983127
Accuratezza del secondo modello:  0.8069531663766362
Intervallo di confidenza:  (-0.0036060269292861394, 0.009810157972639295)
```

- La differenza non è statisticamente significativa
  - L'uso di più variabili non porta benefici apprezzabili

1.5.4 Miglior modello

- Sono stati creati diversi modelli oltre a `model_1`
  - Verifichiamo se fra quelli ottenuti se ne cela uno migliore
  - Vengono confrontati in ordine, dal primo all'ultimo creato

[480]:
```
valuate_complete_model(model_1, X_val_v2, y_val_v2, model, X_val, y_val)
```

```
Accuratezza del primo modello:  0.8069531663766362
Accuratezza del secondo modello:  0.7946205644246047
Intervallo di confidenza:  (0.0055241662871834355, 0.01914103761687947)
```

[481]:
```
valuate_complete_model(model_1, X_val_v2, y_val_v2, random, X_val, y_val)
```

```
Accuratezza del primo modello:  0.8069531663766362
Accuratezza del secondo modello:  0.49708708481501096
Intervallo di confidenza:  (0.30218703511275036, 0.31754512801050006)
```

[482]:
```
valuate_complete_model(model_1, X_val_v2, y_val_v2, model_stand, X_val, y_val)
```

```
Accuratezza del primo modello:  0.8069531663766362
Accuratezza del secondo modello:  0.8097904214269501
Intervallo di confidenza:  (-0.0038726151291398466, 0.009547125229767764)
```

[483]:
```
valuate_complete_model(model_1, X_val_v2, y_val_v2, model_reg_1, X_val, y_val)
```

```
Accuratezza del primo modello:  0.8069531663766362
Accuratezza del secondo modello:  0.8100552318983127
Intervallo di confidenza:  (-0.0036060269292861394, 0.009810157972639295)
```

[484]:
```
valuate_complete_model(model_1, X_val_v2, y_val_v2, model_reg_2, X_val, y_val)
```

```
Accuratezza del primo modello:  0.8069531663766362
Accuratezza del secondo modello:  0.7961337671181055
Intervallo di confidenza:  (0.004020510542159647, 0.01761828797490176)
```

[485]:
```
valuate_complete_model(model_1, X_val_v2, y_val_v2, model_no_deposit,␣
↪X_val_no_deposit, y_val)
```

```
Accuratezza del primo modello:  0.8069531663766362
Accuratezza del secondo modello:  0.8001815843232201
Intervallo di confidenza:  (-1.4609502655225232e-06, 0.013544625057097729)
```

[486]:
```
valuate_complete_model(model_1, X_val_v2, y_val_v2, model_stand_v2, X_val_v2,␣
↪y_val_v2)
```

```
Accuratezza del primo modello:  0.8069531663766362
Accuratezza del secondo modello:  0.8071801467806613
Intervallo di confidenza:  (-0.006500306903338989, 0.0069542677113892575)
```

[487]:
```
valuate_complete_model(model_1, X_val_v2, y_val_v2, gs_poly, X_val_poly, y_val)
```

```
Accuratezza del primo modello:  0.8069531663766362
Accuratezza del secondo modello:  0.8052508133464478
Intervallo di confidenza:  (-0.005037684459347379, 0.008442390519724174)
```

- Nessuno si è dimostrato migliore in modo statisticamente significativo di `model_1`
- Visualizziamo in dettaglio le metriche dei modelli che non si sono dimostrati peggiori in modo statisticamente significativo
  - Ridefiniamo la funzione `print_model_informations` al fine di aggiungerci l'intervallo di confidenza

[488]:
```
def print_model_informations(model,  X_train, y_train, X_val, y_val):
    y_pred = model.predict(X_val)
    print("Accuracy =", model.score(X_val, y_val),"        ( Accuracy on␣
↪training set =", model.score(X_train, y_train),")")
    print("\nIntervallo di condidenza =",model_conf_interval(model, X_val,␣
↪y_val))
    print("\nPrecision (Y) =", precision_score(y_val,y_pred, pos_label="Y"))
    print("Precision (N) =", precision_score(y_val,y_pred, pos_label="N"))
    print("Precision =", precision_score(y_val,y_pred, average="macro"))
    print("\nRecall (Y) =", recall_score(y_val,y_pred, pos_label="Y"))
    print("Recall (N) =", recall_score(y_val,y_pred, pos_label="N"))
    print("Recall =", recall_score(y_val,y_pred, average="macro"))
    print("\nF1 Score (Y) =", f1_score(y_val,y_pred, pos_label="Y"))
    print("F1 Score (N) =", f1_score(y_val,y_pred, pos_label="N"))
    print("F1 Score =", f1_score(y_val,y_pred, average="macro"))
    print("\nMatrice di confusione:")
    cm = confusion_matrix(y_val, y_pred)
```

```
    print(pd.DataFrame(cm, index=model.classes_, columns=model.classes_))
```

[489]:
```
print_model_informations(model_1, X_train_v2, y_train_v2, X_val_v2, y_val_v2)
```

```
Accuracy = 0.8069531663766362         ( Accuracy on training set =
0.8044148523653697 )

Intervallo di condidenza = (0.8021507272563478, 0.8116664039673401)

Precision (Y) = 0.837948252383114
Precision (N) = 0.7914538644875724
Precision = 0.8147010584353431

Recall (Y) = 0.6676914730084095
Recall (N) = 0.9071219512195122
Recall = 0.7874067121139608

F1 Score (Y) = 0.7431935987116906
F1 Score (N) = 0.8453495772342942
F1 Score = 0.7942715879729924

Matrice di confusione:
       N     Y
N  13947  1428
Y   3675  7384
```

[490]:
```
print_model_informations(model_stand, X_train, y_train, X_val, y_val)
```

```
Accuracy = 0.8097904214269501         ( Accuracy on training set =
0.8101462159759396 )

Intervallo di condidenza = (0.80501436141789, 0.814476455391425)

Precision (Y) = 0.8362136247073252
Precision (N) = 0.7962210134554824
Precision = 0.8162173190814037

Recall (Y) = 0.6781806673297767
Recall (N) = 0.9044552845528455
Recall = 0.7913179759413111

F1 Score (Y) = 0.7489514679448771
F1 Score (N) = 0.8468940316686967
F1 Score = 0.7979227498067869

Matrice di confusione:
       N     Y
```

```
N  13906  1469
Y   3559  7500
```

[491]:
```
print_model_informations(model_reg_1, X_train, y_train, X_val, y_val)
```

```
Accuracy = 0.8100552318983127         ( Accuracy on training set =
0.8101273005844856 )

Intervallo di condidenza = (0.8052816543039875, 0.8147387064933193)

Precision (Y) = 0.8366413916146298
Precision (N) = 0.7964044429176687
Precision = 0.8165229172661492

Recall (Y) = 0.6784519395967086
Recall (N) = 0.9047154471544715
Recall = 0.79158369337559

F1 Score (Y) = 0.7492884605782194
F1 Score (N) = 0.8471118419049358
F1 Score = 0.7982001512415776

Matrice di confusione:
       N     Y
N  13910  1465
Y   3556  7503
```

[492]:
```
print_model_informations(model_no_deposit, X_train_no_deposit, y_train,␣
↪X_val_no_deposit, y_val)
```

```
Accuracy = 0.8001815843232201         ( Accuracy on training set =
0.7994968505873229 )

Intervallo di condidenza = (0.7953177691651377, 0.8049581657941874)

Precision (Y) = 0.7922695537792168
Precision (N) = 0.8049060479729322
Precision = 0.7985878008760745

Recall (Y) = 0.7080206166922868
Recall (N) = 0.8664715447154472
Recall = 0.787246080703867

F1 Score (Y) = 0.7477795817018431
F1 Score (N) = 0.834554908225271
F1 Score = 0.7911672449635571

Matrice di confusione:
```

```
       N     Y
N  13322  2053
Y   3229  7830
```

[493]:
```
print_model_informations(model_stand_v2, X_train_v2, y_train_v2, X_val_v2,␣
↪y_val_v2)
```

```
Accuracy = 0.8071801467806613         ( Accuracy on training set =
0.8045472601055479 )

Intervallo di condidenza = (0.8023798033852944, 0.8118912226852438)

Precision (Y) = 0.8381352087114338
Precision (N) = 0.7916903167215348
Precision = 0.8149127627164843

Recall (Y) = 0.668143593453296
Recall (N) = 0.9071869918699187
Recall = 0.7876652926616073

F1 Score (Y) = 0.7435471698113209
F1 Score (N) = 0.8455126845088352
F1 Score = 0.7945299271600781

Matrice di confusione:
       N     Y
N  13948  1427
Y   3670  7389
```

[494]:
```
print_model_informations(gs_poly, X_train_poly, y_train, X_val_poly, y_val)
```

```
Accuracy = 0.8052508133464478         ( Accuracy on training set =
0.8056821835927894 )

Intervallo di condidenza = (0.8004327365232091, 0.8099801833491027)

Precision (Y) = 0.8265385040327036
Precision (N) = 0.7941667146062245
Precision = 0.810352609319464

Recall (Y) = 0.6764626096392079
Recall (N) = 0.8978861788617886
Recall = 0.7871743942504983

F1 Score (Y) = 0.7440079562406763
F1 Score (N) = 0.8428475486903961
F1 Score = 0.7934277524655362
```

```
Matrice di confusione:
       N     Y
N  13805  1570
Y   3578  7481
```

- Non abbiamo ottenuto nessun modello con una accuratezza più alta di `model_1` che presenta una differenza di accuratezza statisticamente significativa
  - Tutte le metriche sono molto simili, il modello che presenta le metriche leggermente più alte ora potrebbe ottenere metriche peggiori su un altro validation set
    - Mantengo quindi come modello di riferimento `model_1`

1.5.5 Addestramento miglior configurazione su entrambi i dataset

- Per curiosità addestriamo la configurazione migliore su tutto il dataset, comprensivo di entrambi gli hotel
  - Dobbiamo aggiungere tutte le istanze dell'hotel *Resort Hotel* al dataset di training in nostro possesso, contenente ora sole istanze di *City Hotel*
  - Ripetiamo le operazioni effettuate nel paragrafo *Classificazione Lineare* per ottenere la parte del dataset mancante
    - Rimuoviamo le istanze di *City Hotel*
    - Rimuoviamo la variabile *hotel* , in quanto inutile
    - Cambiamo in *Y* e *N* le istanze della variabile da predire
    - Separiamo la variabile da predire dalle altre variabili
    - Binarizziamo tutte le variabili categoriche
      - Manteniamo solo il sottoinsieme di variabili utilizzato nei modelli precedenti

[495]:
```
hbd_resort = hbd_complete[hbd_complete["hotel"] == "Resort Hotel"].copy()
hbd_resort.drop(inplace=True, axis=1, labels=['hotel'])
hbd_resort["is_canceled"] = hbd_resort["is_canceled"].map(lambda value: "N" if␣
↪value is 0 else "Y")
y_resortHotel = hbd_resort["is_canceled"]
X_resortHotel = hbd_resort.drop(columns="is_canceled")
X_resortHotel = pd.get_dummies(X_resortHotel)
X_resortHotel = X_resortHotel[X_train_v2.columns]
```

- Prendo le variabili `X_resortHotel` e `y_resortHotel` definite, che contengono tutte le istanze del dataset *Resort Hotel* già private di tutte le variabili meno rilevanti, e le unisco rispettivamente a `X_train_v2` e `y_train_v2`
  - In questo modo mantengo un validation set identico ai test precedenti, e al tempo stesso aggiungo le istanze del `Resort_Hotel` al training set
    - Il nostro obiettivo è sempre realizzare un modello per *City Hotel* , dunque il nostro validation set non deve comprendere istanze di *Resort Hotel*

[496]:
```
y_train_all = y_train_v2.append(y_resortHotel)
X_train_all = X_train_v2.append(X_resortHotel)
```

- Riporto la configurazione del primo modello

[497]:
```
model_1_both_dataset = copy.deepcopy(gs.best_estimator_.set_params(**result.
↪loc[0, 'params']))
```

- Lo addesto sul dataset...

[498]:
```
%time model_1_both_dataset.fit(X_train_all, y_train_all)
```

```
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_logistic.py:1501: UserWarning: l1_ratio parameter
is only used when penalty is 'elasticnet'. Got (penalty=l1)
  "(penalty={})".format(self.penalty))
```

```
Wall time: 9.62 s
```

```
C:\Users\alessandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

[498]:
```
Pipeline(memory=None,
         steps=[('scaler',
                 StandardScaler(copy=True, with_mean=True, with_std=True)),
                ('lr',
                 LogisticRegression(C=1.0, class_weight=None, dual=False,
                                    fit_intercept=True, intercept_scaling=1,
                                    l1_ratio=0.5, max_iter=100,
                                    multi_class='auto', n_jobs=None,
                                    penalty='l1', random_state=42,
                                    solver='saga', tol=0.0001, verbose=0,
                                    warm_start=False))],
         verbose=False)
```

- Visualizziamone le metriche

[499]:
```
print_model_informations(model_1_both_dataset, X_train_all, y_train_all,␣
↪X_val_v2, y_val_v2)
```

```
Accuracy = 0.7995006431111448         ( Accuracy on training set =
0.8095995154818684 )

Intervallo di condidenza = (0.794630777023733, 0.8042834733950416)

Precision (Y) = 0.8404066674547819
Precision (N) = 0.780250347705146
Precision = 0.810328507579964

Recall (Y) = 0.6428248485396509
Recall (N) = 0.9121951219512195
Recall = 0.7775099852454352
```

```
F1 Score (Y) = 0.7284557844041396
F1 Score (N) = 0.8410794602698651
F1 Score = 0.7847676223370024

Matrice di confusione:
       N     Y
N  14025  1350
Y   3950  7109
```

- Confrontiamolo con il modello migliore in nostro possesso, quello addestrato solo su istanze di *City Hotel*

[500]:
```
valuate_complete_model(model_1_both_dataset, X_val_v2, y_val_v2, model_1,␣
↪X_val_v2, y_val_v2)
```

```
Accuratezza del primo modello:  0.7995006431111448
Accuratezza del secondo modello:  0.8069531663766362
Intervallo di confidenza:  (0.0006751009095118986, 0.014229945621470893)
```

- Abbiamo ottenuto un modello con una accuratezza inferiore, la cui differenza con quella del nostro miglior modello risulta essere statisticamente significativa
  - Possiamo quindi affermare che non è conveniente addestrare il modello su tutte le istanze se questo è destinato alla sola struttura *City Hotel*

1.6 Interpretazione della conoscenza appresa

- Interpretiamo ora la conoscenza appresa attraverso l'analisi dei parametri (o coefficienti degli iperpiani) appresi
- Analizziamo quali feature sono più positivamente o negativamente correlate ed in che misura con la variabile da predire

[501]:
```
model_1
```

[501]:
```
Pipeline(memory=None,
         steps=[('scaler',
                 StandardScaler(copy=True, with_mean=True, with_std=True)),
                ('lr',
                 LogisticRegression(C=1.0, class_weight=None, dual=False,
                                    fit_intercept=True, intercept_scaling=1,
                                    l1_ratio=None, max_iter=100,
                                    multi_class='auto', n_jobs=None,
                                    penalty='l1', random_state=42,
                                    solver='saga', tol=0.0001, verbose=0,
                                    warm_start=False))],
         verbose=False)
```

- Visualizziamo prima di tutto l'intercetta di questo modello

[502]:
```
model_1.named_steps["lr"].intercept_
```

[502]:
```
array([0.05525542])
```

- La probabilità di partenza è molto vicina a 0 (ovvero la classe N)
- Mostriamo tutti i coefficienti ordinati in ordine decrescente (non in valore assoluto)

[503]:
```
coeff = pd.Series(model_1.named_steps["lr"].coef_[0], index=X_train_v2.columns)
coeff.sort_values(ascending=False)
```

[503]:
```
deposit_type_Non Refund           1.268058
previous_cancellations            0.696396
market_segment_Online TA          0.628510
country_PRT                       0.621384
lead_time                         0.559141
customer_type_Transient           0.257421
adr                               0.241440
stays_in_week_nights              0.172971
country_CHN                       0.096069
country_AGO                       0.087563
stays_in_weekend_nights           0.082684
country_ITA                       0.076589
meal_FB                           0.068886
country_ARE                       0.064308
meal_SC                           0.062647
distribution_channel_TA/TO        0.055978
country_HKG                       0.055726
adults                            0.055303
country_BRA                       0.054443
arrival_date_day_Saturday         0.045202
customer_type_Transient-Party     0.015836
company                          -0.058547
country_CHE                      -0.062180
country_SWE                      -0.067080
country_BEL                      -0.071994
country_NLD                      -0.084534
country_AUT                      -0.089240
market_segment_Offline TA/TO     -0.110532
country_FRA                      -0.161704
country_DEU                      -0.223984
booking_changes                  -0.240963
total_of_special_requests        -0.643048
previous_bookings_not_canceled   -0.717954
deposit_type_No Deposit          -1.062401
required_car_parking_spaces      -1.370483
dtype: float64
```

- Analizzando i coefficienti possiamo ritrovare tutte le considerazioni fatte in fase di analisi esplorativa

- Facciamo dunque notare solo qualche dettaglio:
  - La variabili relative alle diverse nazioni ricoprono, per la maggior parte, un ruolo non di primo piano
    * Non tutte sono state annullate in quanto, evidentemente, alcune hanno rilevanza per effettuare una predizione
    * Tra queste la variabile *PRT* assume un ruolo di rilievo
      · Si era visto in fase esplorativa che la variabile `country` presentava una alto tasso di cancellazioni da parte dei clienti provenienti dal Portogallo (ovvero la maggior parte essendo l'hotel portoghese)
      · Il modello ha appreso questa informazione, aumentando di molto la probabilità che la prenotazione venga cancellata se questa è effettuata da un cliente portoghese
    * Esistono anche molte nazionalità che il modello ha riconosciuto come poco propense a cancellare le proprie prenotazioni
  - Le variabili `deposit_type_No Deposit` e `deposit_type_Non Refund` assumono anche esse un ruolo di particolare rilievo
    * `deposit_type` con valore *Non Refund* aumenta la probabilità che la prenotazione venga considerata come cancellata
    * `deposit_type` con valore *No Deposit* , che presentava a differenza di *Non Refund* una bassa percentuale di cancellazioni, agisce esattamente nel senso opposto
  - La variabile `marker_segment` quando assume il valore *Online TA* contribuisce in modo rilevante nel valutare la prenotazione come cancellabile
  - Avevamo visto che la variabile `required_car_parking_spaces` quando maggiore di 0 presentava zero cancellazioni, e dai coefficienti apprendiamo che tale conoscenza è tenuta in cosiderazione dal modello
    * La probabilità che una prenotazione venga cancellata tende a diminuire tante più sono le macchine
- Notiamo che la probabilità che una prenotazione venga cancellata dimuisce…
  - tanto più sono le richieste speciali
  - tanto più sono le prenotazioni non cancellate
  - tanto più sono le modifiche alla prenotazione
- Notiamo che la probabilità che una prenotazione venga cancellata aumenta…
  - tanto più sono le prenotazione cancellate
  - tanto più è il tempo che intercorre dalla prenotazione all'arrivo in hotel
  - tanto più è il costo medio per notte
- Il resto delle variabili può essere intepretato nel medesimo modo

1.7 (Plus) Classificazione con reti neurali

- Quello che possiamo fare ora è creare un modello di classificazione di tipo *multi-layer perceptron*
- Per creare un modello di classificazione di questo tipo usiamo la classe `MLPClassifier`

  - con `hidden_layer_sizes` specifichiamo il numero di variabili nascoste da introdurre
  - con `activation="identity"` specifichiamo che tali variabili sono lineari

- Addestriamo il modello su tutte le istanze dell'hotel *City Hotel* , con tutte le variabili ottenute dalla binarizzazione
- Per prima cosa realizziamo un modello con sole funzioni di attivazione *identity*

[504]:
```
from sklearn.neural_network import MLPClassifier
```

[505]:
```
model_linear = MLPClassifier(hidden_layer_sizes=(16,8,4,2),␣
 ↪activation="identity")
%time model_linear.fit(X_train, y_train)
print_model_informations(model_linear,  X_train, y_train, X_val, y_val)
```

```
Wall time: 15.7 s
Accuracy = 0.8079745781947492         ( Accuracy on training set =
0.8082357614390829 )

Intervallo di condidenza = (0.803181589765629, 0.8126780682688847)

Precision (Y) = 0.8595120778752554
Precision (N) = 0.7842985700877823
Precision = 0.8219053239815188

Recall (Y) = 0.646713084365675
Recall (N) = 0.9239674796747968
Recall = 0.7853402820202359

F1 Score (Y) = 0.7380804953560371
F1 Score (N) = 0.8484233158146202
F1 Score = 0.7932519055853287

Matrice di confusione:
       N     Y
N  14206  1169
Y   3907  7152
```

- Questo modello è quasi identico al nostro miglior modello, confrontiamolo...

[506]:
```
valuate_complete_model(model_linear, X_val, y_val, model_1, X_val_v2, y_val_v2)
```

```
Accuratezza del primo modello:  0.8079745781947492
Accuratezza del secondo modello:  0.8069531663766362
Intervallo di confidenza:  (-0.005700594994948269, 0.007743418631174257)
```

- Poichè l'intervallo ottenuto include lo zero (l'estremo inferiore è negativo), non abbiamo la certezza al 95% che il modello con accuratezza stimata maggiore sia effettivamente migliore
  - La differenza non è statisticamente significativa

- Proviamo a fare di meglio
- L'output finale è anche qui una combinazione lineare dell'input
- Possiamo aggiungere espressività al modello introducendo trasformazioni non lineari
  - La funzione *ReLU* (*rectified linear unit*) è un esempio di funzione che introduce non linearità
- Le reti neurali durante il loro addestramento identificano, di fatto, una funzione Kernel.
  - Non dovendo calcolare alcuna feature non abbiamo alcun problema ad applicare trasformazioni non lineari al dataset completo

[507]:
```
model_not_linear = MLPClassifier(hidden_layer_sizes=8, batch_size=50,␣
↪activation="relu", random_state=42)
%time model_not_linear.fit(X_train, y_train)
print_model_informations(model_not_linear, X_train, y_train,  X_val, y_val)
```

```
Wall time: 1min 35s
Accuracy = 0.8409623969130665         ( Accuracy on training set =
0.8456314903436927 )

Intervallo di condidenza = (0.8365042543388092, 0.8453214547712599)

Precision (Y) = 0.8201176800224153
Precision (N) = 0.855153557576143
Precision = 0.8376356187992791

Recall (Y) = 0.7940139253097025
Recall (N) = 0.8747317073170732
Recall = 0.8343728163133879

F1 Score (Y) = 0.8068547275567399
F1 Score (N) = 0.8648318436113434
F1 Score = 0.8358432855840416

Matrice di confusione:
       N     Y
N  13449  1926
Y   2278  8781
```

- Confrontiamolo ora con il nostro migliore modello senza rete neurale

[508]:
```
print_model_informations(model_1, X_train_v2, y_train_v2, X_val_v2, y_val_v2)
```

```
Accuracy = 0.8069531663766362         ( Accuracy on training set =
0.8044148523653697 )

Intervallo di condidenza = (0.8021507272563478, 0.8116664039673401)
```

```
Precision (Y) = 0.837948252383114
Precision (N) = 0.7914538644875724
Precision = 0.8147010584353431

Recall (Y) = 0.6676914730084095
Recall (N) = 0.9071219512195122
Recall = 0.7874067121139608

F1 Score (Y) = 0.7431935987116906
F1 Score (N) = 0.8453495772342942
F1 Score = 0.7942715879729924

Matrice di confusione:
       N     Y
N  13947  1428
Y   3675  7384
```

- Applichiamo il confronto sull'accuratezza

[509]:
```
valuate_complete_model(model_not_linear, X_val, y_val, model_1, X_val_v2,␣
↪y_val_v2)
```

```
Accuratezza del primo modello:  0.8409623969130665
Accuratezza del secondo modello:  0.8069531663766362
Intervallo di confidenza:  (0.027522748323993533, 0.04049571274886716)
```

- Poichè l'intervallo ottenuto non include lo zero (l'estremo inferiore è positivio), abbiamo la certezza al 95% che il modello con accuratezza stimata maggiore sia effettivamente migliore
  - La differenza è statisticamente significativa

- Mostriamo coefficienti e intercette

[510]:
```
model_not_linear.coefs_
```

[510]:
```
[array([[ 7.18113655e-03,  6.55512857e-03,  4.43700556e-02, …,
         -3.26398515e-01, -8.56035662e-02,  2.70048887e-03],
        [ 3.65784384e-02,  1.20535844e-01,  9.81561800e-02, …,
         -6.91123771e-03,  9.42045373e-02, -9.18279540e-03],
        [-4.81001105e-01, -1.51023713e-01,  1.83385681e-01, …,
         -1.90153697e-01,  3.05962924e-01, -4.03060257e-02],
        …,
        [ 2.33676861e-01,  7.71001692e-01, -1.18282363e-01, …,
         -1.72480991e-01,  4.59267783e-01,  3.53492206e+00],
        [ 1.00070180e+00, -7.53483986e-01, -7.34250157e-02, …,
          2.65901243e-01,  7.96848440e-01, -4.76132999e-01],
        [-2.82783910e+00,  2.53973015e+00, -1.72098236e+00, …,
         -2.68042324e-01, -1.74190402e+00, -3.18200508e-03]]),
 array([[ 0.33065989],
        [-0.28393118],
```

```
           [ 0.19851285],
           [ 0.37427723],
           [ 0.34274162],
           [-0.61334057],
           [-0.30045459],
           [-1.59868305]])]
```

[511]:
```
model_not_linear.intercepts_
```

[511]:
```
[array([-0.18476474,  0.33361153, -0.374455  ,  0.31003749,  0.34980676,
         0.29620993,  0.37467228, -0.21018972]),
 array([-0.56713692])]
```

- Con la funzione *ReLu* abbiamo ottenuto il miglior modello in nostro possesso
  - Osservando i coefficienti posso però affermare di non essere in grado di comprendere su cosa il modello si basi per trovare il risultato
  - Se volessi adottare questo modello rispetto dovrei rinunciare alla possibilità di sapere su cosa le nostre decisioni si basano