

hotel-bookings

June 13, 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

```
[464]: # 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

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

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

```
[466]:      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 oppure sostituiti con identificativi anonimi
- Il dataset presenta le seguenti dimensioni...

```
[467]: 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 Lunedì a Venerdì) 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 strategie 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: Sì, 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 prenotazioni 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**: Average 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 cliente 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

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

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

```
[469]:
```

	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

1.2 Preparazione dei dati

- Riportiamo lo spazio occupato in memoria dal dataset

```
[470]: 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

```
[471]: 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

```
[472]: 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 specifica struttura ospitante, la struttura *City Hotel*
 - Il dataset prevede istanze di due strutture: *Resort Hotel* e *City Hotel*

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

```
[473]: [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

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

```
[474]: 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
- Le 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 (Lunedì, Martedì, ...)
 - 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. Febbraio, ...)

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

- Mostriamo un esempio

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

```
[476]: 0    July
Name: arrival_date_month, dtype: category
Categories (12, object): [April, August, December, February, ..., May, November,
October, September]
```

```
[477]: hbd_complete["arrival_date_month"].head(1).map(dict_month_conversion)
```

```
[477]: 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`

```
[478]: arrival_date_month_number = hbd_complete["arrival_date_month"].
↳map(dict_month_conversion)
```

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

```
[479]: 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

```
[480]: 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

```
[481]: 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

```
[482]: 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...

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

```
[483]: [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.
 - Definiamo 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)

```
[484]: 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

```
[485]: reservation_status_mapped = hbd_complete["reservation_status"].
↳map(mapping_reservation_status)
```

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

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

```
[486]: 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`

```
[487]: hbd_complete.drop(inplace=True, axis=1, labels=['reservation_status',
↳'reservation_status_date'])
```

- Verifichiamo la presenza del valore `nan` nelle istanze

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

```
[488]: 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

```
[489]: 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 presenza (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

```
[490]: 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`

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

```
[491]: 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

```
[492]: hbd_complete.dtypes
```

```
[492]: 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

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

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

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

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

- Il dataset ora presenta...

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

79302 istanze

- La variabile hotel è diventata inutile, è quindi rimossa

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

1.3 Analisi esplorativa

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

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

```
[498]:
```

	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 (\sim) 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

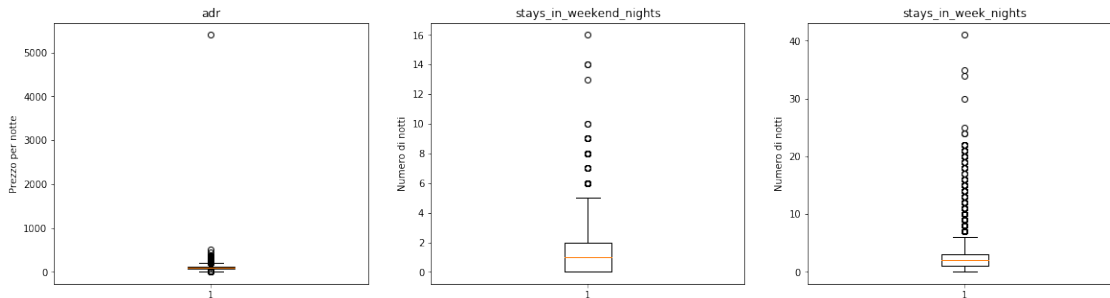
```
[499]: plt.figure(figsize=(20, 5))

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 lontano dalla media
 - * Viene rimosso in quanto potrebbe potenzialmente compromettere i grafici dell'analisi esplorativa

```
[500]: 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

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

```
[501]: <pandas.io.formats.style.Styler at 0x29212227608>
```

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

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

```
[502]: 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

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

```
[503]: 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...

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

```
[504]: 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...

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

```
[505]: 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

```
[506]: 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")
```

```
[506]: <seaborn.axisgrid.PairGrid at 0x2921b666188>
```



- 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 né cancellazioni né 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
- Analizziamo alcune variabili più nel dettaglio, iniziando dalla variabile dipendente

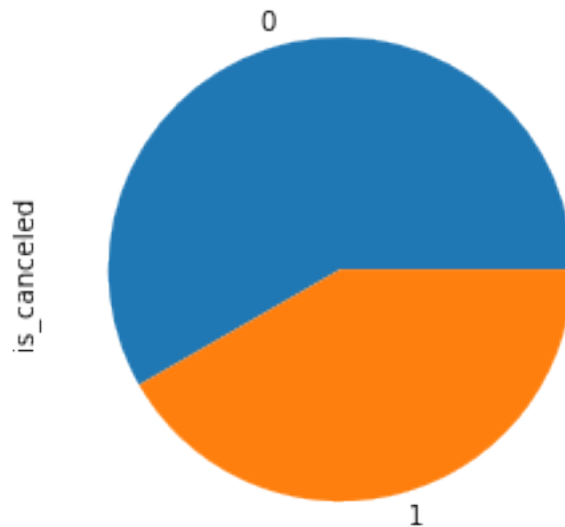
1.3.1 Variabile **is_canceled**

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

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

```
[507]: 0    46226  
      1    33075  
      Name: is_canceled, dtype: int64
```

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



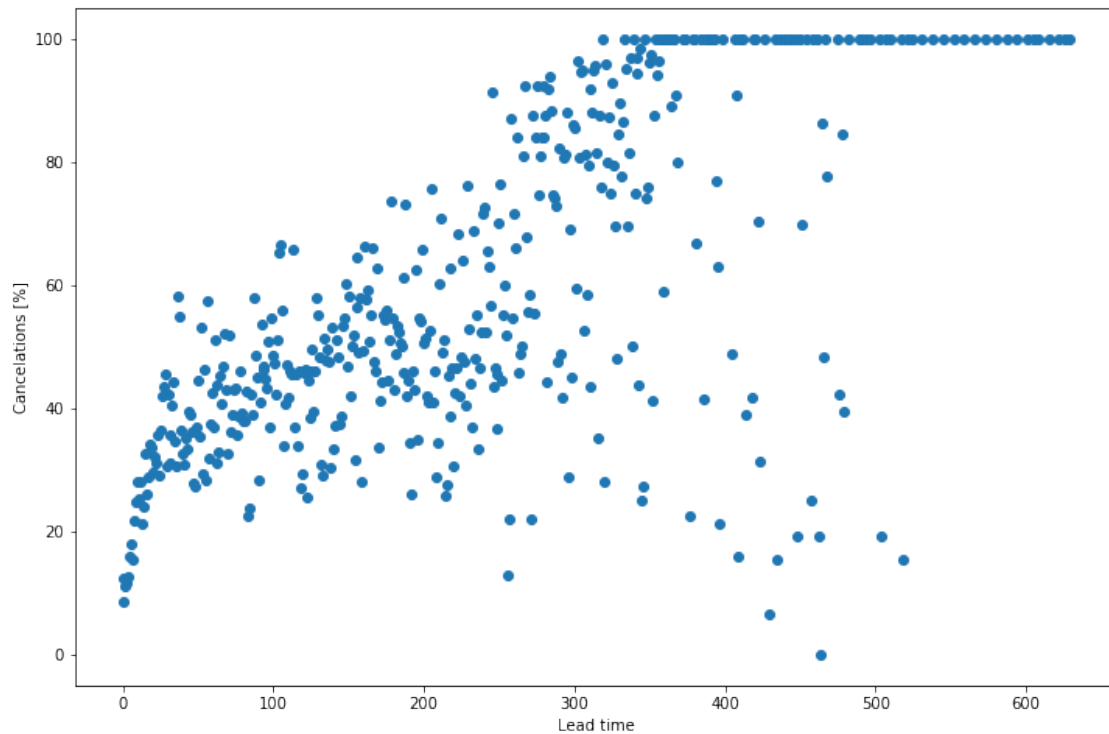
- La suddivisione delle istanze nelle classi è abbastanza bilanciata, non siamo dunque soggetti ai problemi che un forte sbilanciamento comporterebbe.
- Analizziamo ora le variabili che hanno ottenuto il maggiore valore di correlazione con la variabile dipendente, ovvero:
 - lead_time
 - total_of_special_requests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

```
[509]: 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 [%]")
```

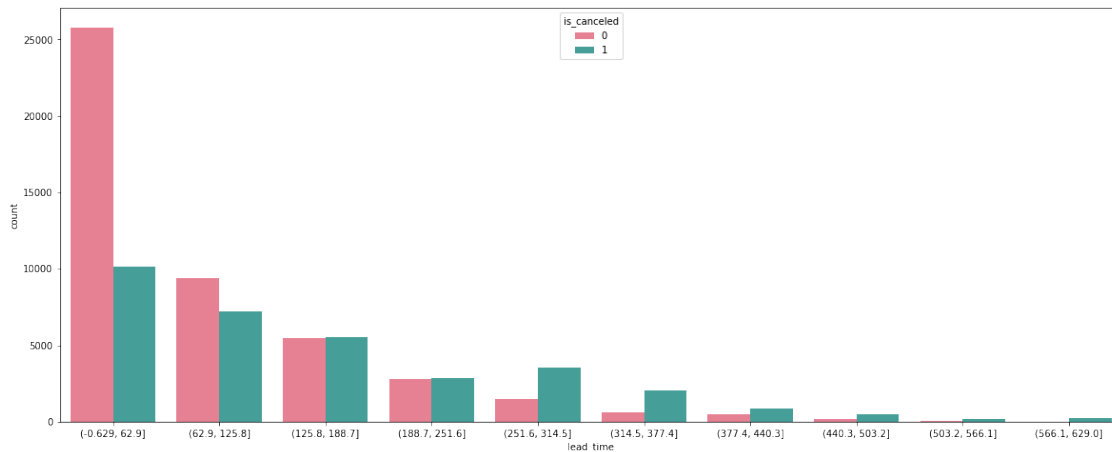
```
[509]: Text(0, 0.5, 'Cancellations [%]')
```



- Le prenotazioni 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

```
[510]: 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")
```

```
[510]: <matplotlib.axes._subplots.AxesSubplot at 0x29212284d88>
```



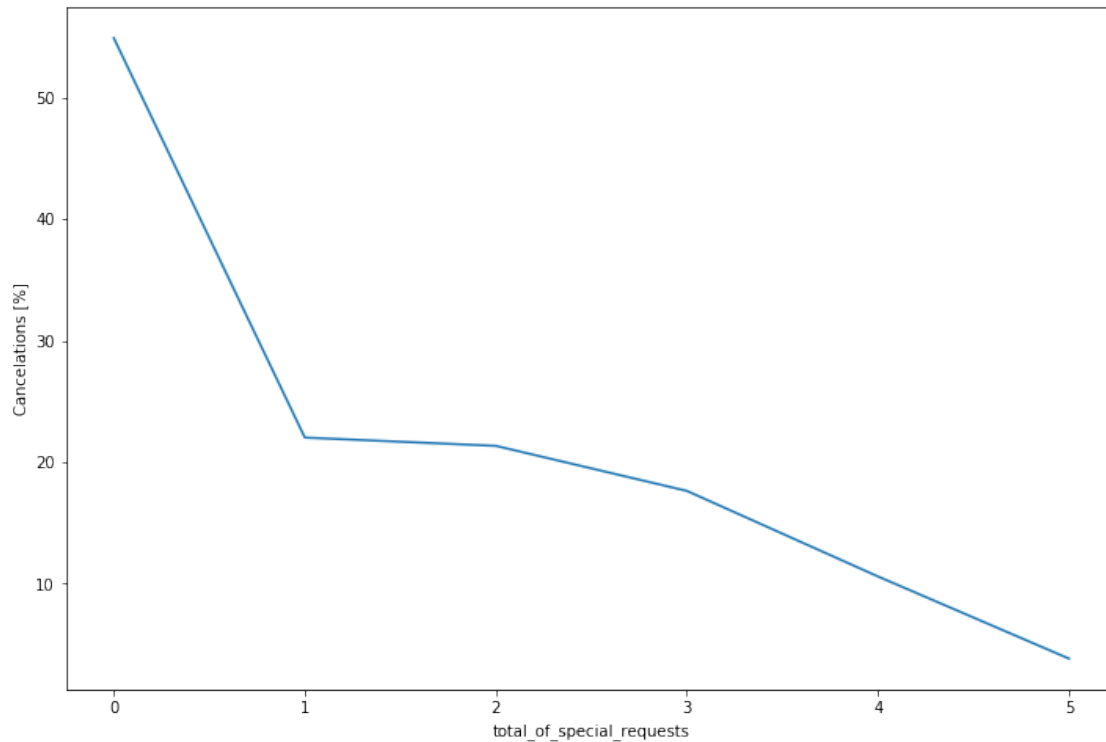
- Le prenotazioni più frequenti sono quelle a breve termine, ovvero senza lunghi periodi fra il momento della prenotazione e l'arrivo in hotel
- Medesima cosa vale per le cancellazioni, anche esse maggiori nelle prenotazioni a breve termine

1.3.3 Variabile total_of_special_requests

- Visualizziamo la percentuale di cancellazione per i valori di total_of_special_requests

```
[511]: 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()
```

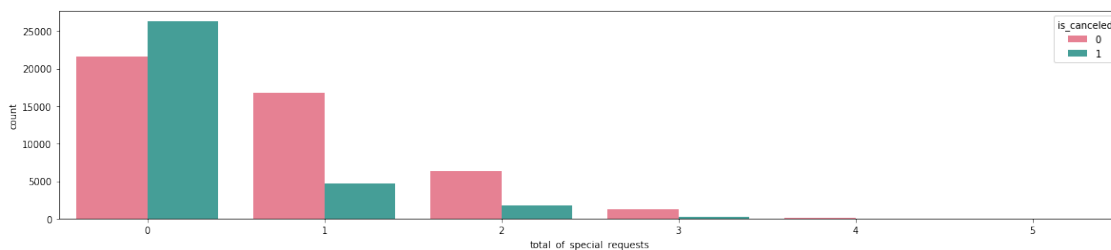
```
[511]: <matplotlib.axes._subplots.AxesSubplot at 0x2922cb05588>
```



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

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

[512]: <matplotlib.axes._subplots.AxesSubplot at 0x2923179cdc8>



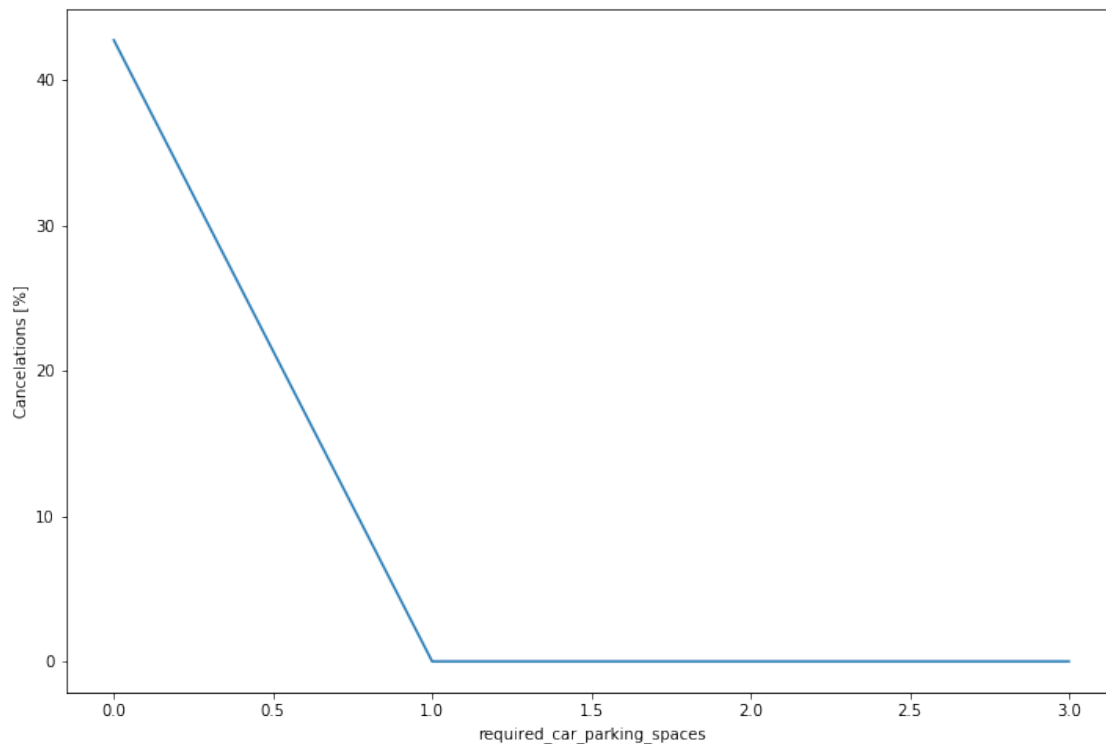
- All'aumentare del numero di richieste speciali la percentuale di cancellazioni diminuisce
 - Tanto più un cliente effettua richieste speciali tanto più è raro che cancelli la sua prenotazione

1.3.4 Variabile required_car_parking_spaces

- Visualizziamo la percentuale di cancellazione per i valori di required_car_parking_spaces

```
[513]: 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()
```

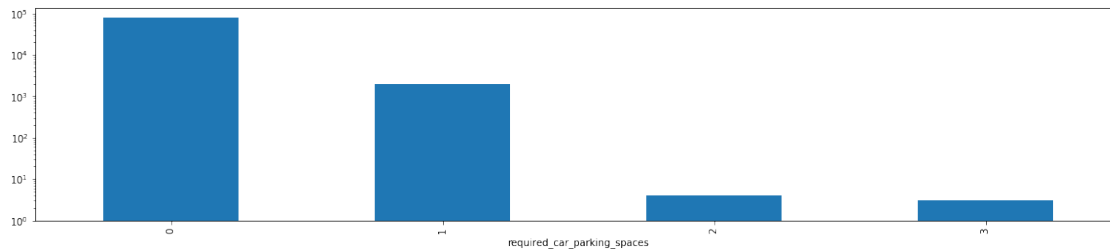
```
[513]: <matplotlib.axes._subplots.AxesSubplot at 0x2923a616608>
```



- Visualizziammo anche il numero di prenotazioni effettuate per ciascun valore di required_car_parking_spaces

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

```
[514]: <matplotlib.axes._subplots.AxesSubplot at 0x2923db2dc88>
```

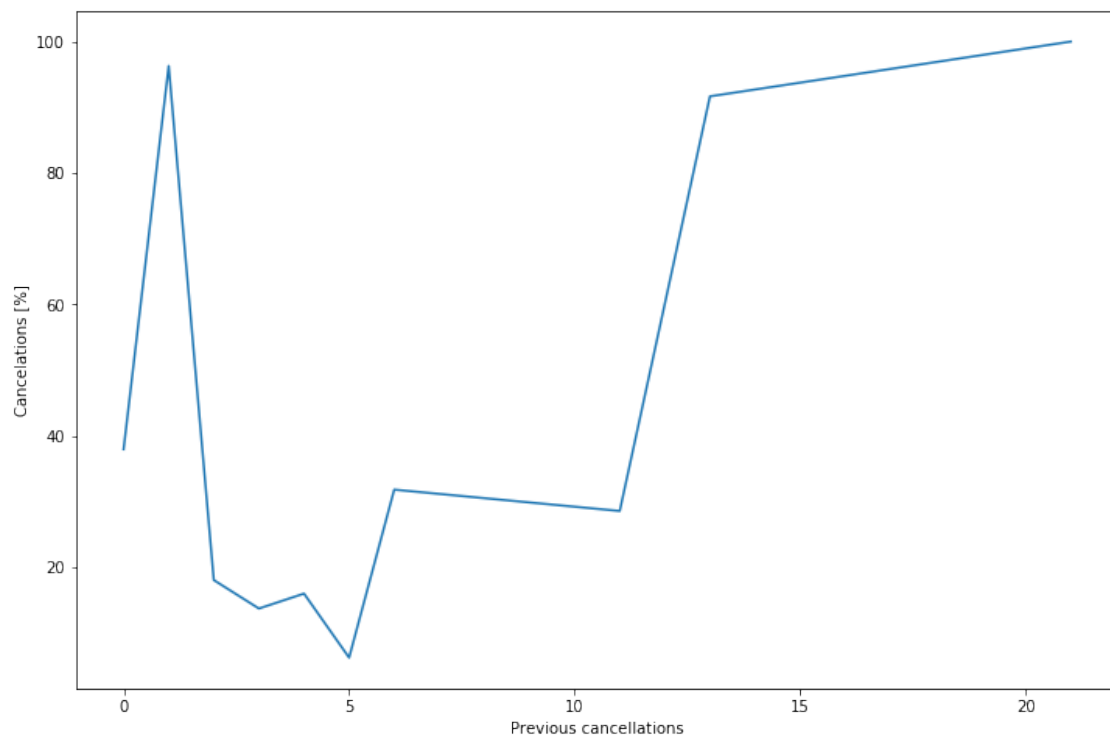
- Nessuna prenotazione per la quale era previsto almeno un posto macchina è mai stata cancellata
 - Chi viene in macchina in genere non cancella la propria prenotazione

1.3.5 Variabile previous_cancellations

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

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

[515]: <matplotlib.axes._subplots.AxesSubplot at 0x292329b7288>



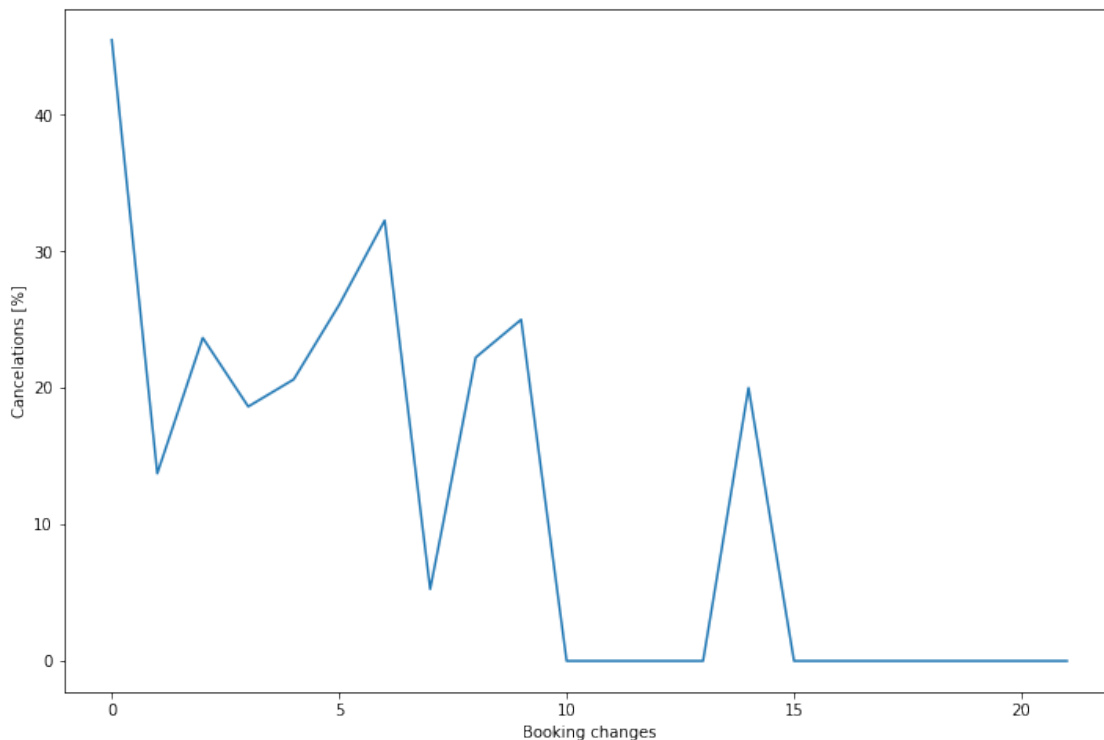
- Eccetto inizialmente, all'aumentare del numero di prenotazioni già cancellate sembra essere sempre più frequente un'ulteriore cancellazione
 - Chi ha già cancellato molte prenotazioni probabilmente lo rifarà (tanto più quante sono le cancellazioni fatte)

1.3.6 Variabile booking_changes

- Visualizziamo la percentuale di cancellazione per i valori di booking_changes

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

```
[516]: <matplotlib.axes._subplots.AxesSubplot at 0x29231f16c88>
```

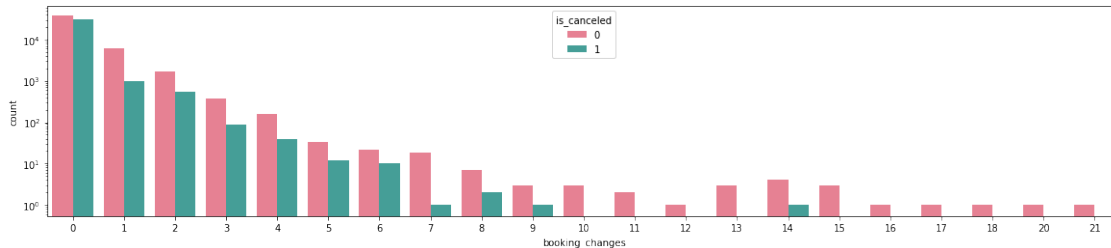


- Visualizziamo un l'istogramma in cui andiamo a quantificare la frequenza delle classi nei valori che la variabile assume

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

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

```
[517]: <matplotlib.axes._subplots.AxesSubplot at 0x29231e18a48>
```



- 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

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

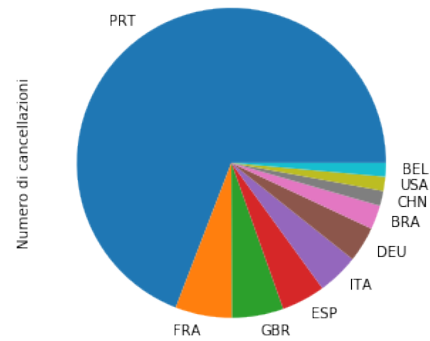
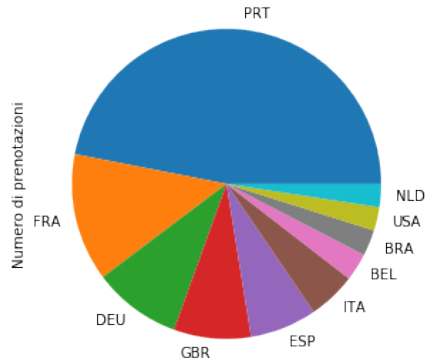
- Memorizziamo i primi 10 stati per numero di prenotazioni

```
[519]: 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

```
[520]: 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")
```

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



- Lo stato che predomina 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 stati

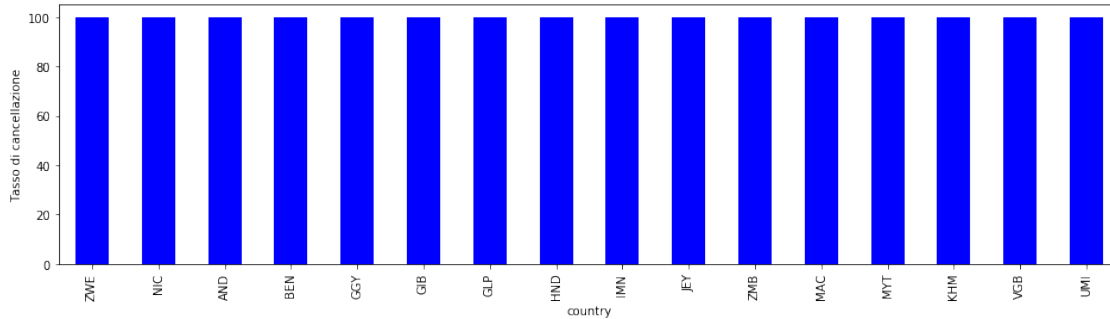
```
[521]: ratio_cancellation_by_state = (100* cancellation_by_state /
↳reservation_by_state).sort_values(ascending=False)
```

- Mostriamo 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*

```
[522]: 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"
```

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

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

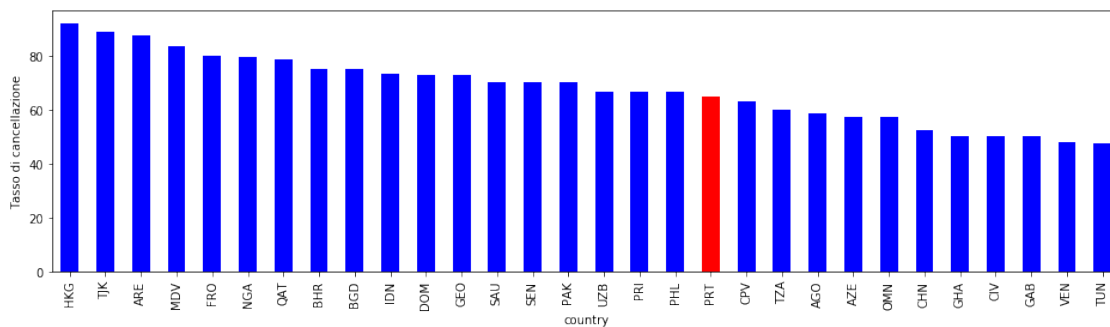


- Nessuno stato con un gran numero di prenotazioni ha percentuale di cancellazione pari a 100%
- Proviamo ora con i 30 stati successivi, replicando lo stesso approccio

```
[524]: 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"
```

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

```
[525]: 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

```
[526]: ratio_cancellation_by_state["PRT"]
```

```
[526]: 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

```
[527]: 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")
```

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



- Il valore `Refundable` è praticamente inesistente

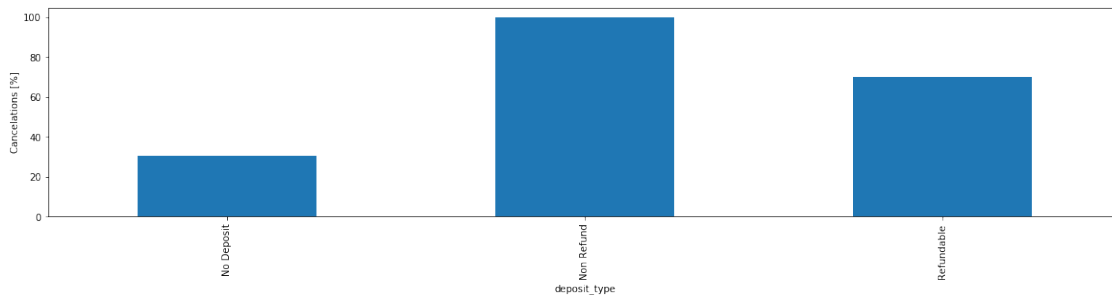
```
[528]: hbd["deposit_type"].value_counts()
```

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

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

```
[529]: (100 * hbd.groupby("deposit_type").sum()["is_canceled"] / hbd["deposit_type"].
        ↳value_counts()).plot.bar()
plt.ylabel("Cancelations [%]")
```

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



- Visualizziamo i valori esatti

```
[530]: (hbd.groupby("deposit_type").sum()["is_canceled"] / hbd["deposit_type"].
        ↳value_counts())
```

```
[530]: 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

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

```
[531]: 1    12829
0         24
Name: is_canceled, dtype: int64
```

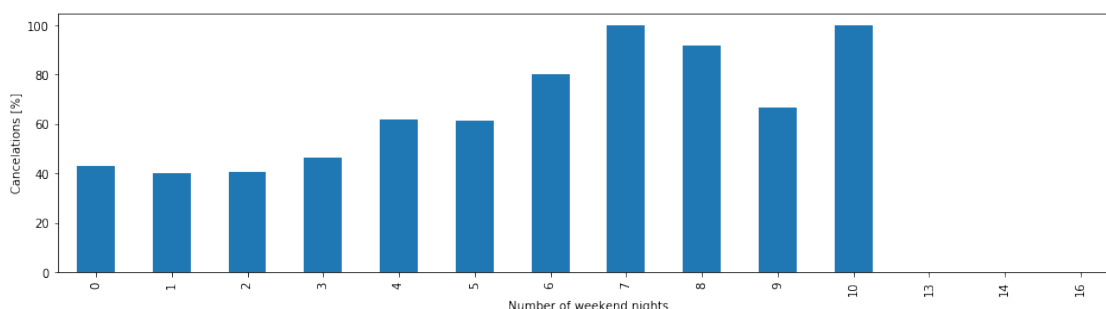
- Effettivamente pare che quasi tutte le prenotazioni *Non Refund* siano state cancellate
 - Poichè pare un fatto particolarmente strano, sarà opportuno verificare in modo attento quanto questa variabile incida sul risultato

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

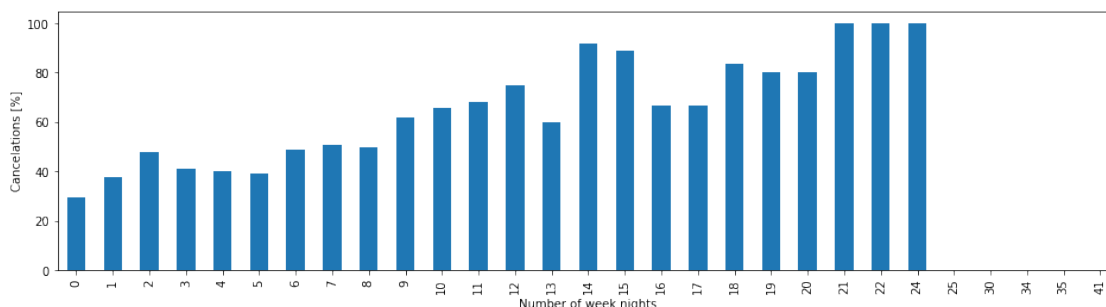
```
[532]: (100 * hbd.groupby("stays_in_weekend_nights").sum()["is_canceled"] /  
        ↳hbd["stays_in_weekend_nights"].value_counts()).plot.bar(figsize=(16,4))  
plt.ylabel("Cancellations [%]")  
plt.xlabel("Number of weekend nights")
```

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



```
[533]: (100 * hbd.groupby("stays_in_week_nights").sum()["is_canceled"] /  
        ↳hbd["stays_in_week_nights"].value_counts()).plot.bar(figsize=(16,4))  
plt.ylabel("Cancellations [%]")  
plt.xlabel("Number of week nights")
```

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



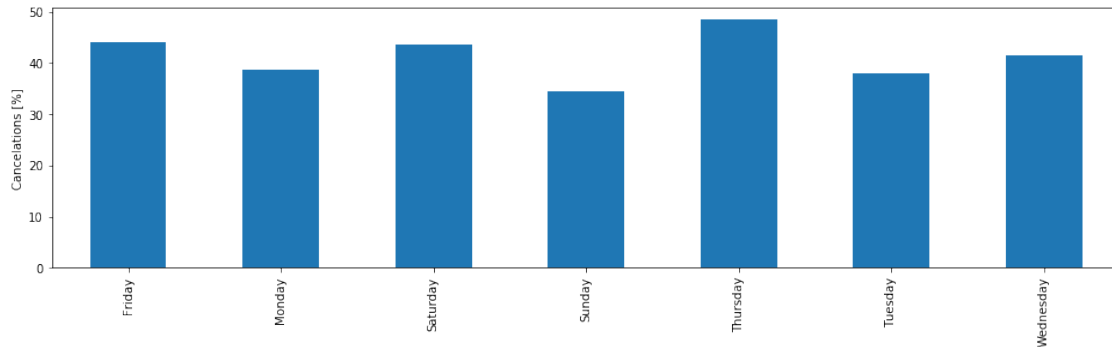
- In entrambi i casi possiamo notare all'aumentare del numero di notti prenotate i clienti tendono a cancellare le prenotazioni con più frequenza
 - Vengono raggiunti picchi pari al 100%

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

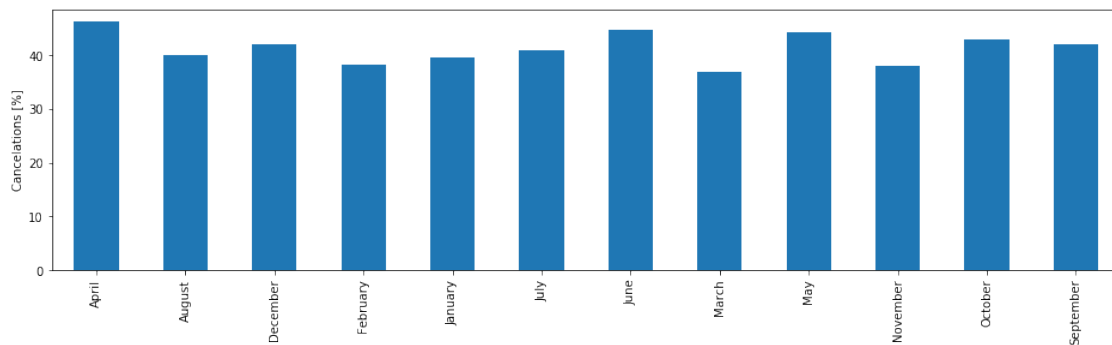
```
[534]: (100 * hbd.groupby("arrival_date_day").sum()["is_canceled"] /  
        ↳hbd["arrival_date_day"].value_counts()).plot.bar(figsize=(16,4))  
plt.ylabel("Cancelations [%]")
```

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



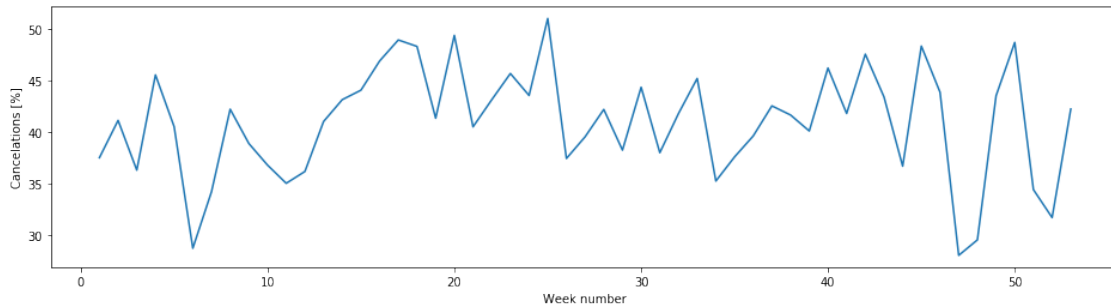
```
[535]: (100 * hbd.groupby("arrival_date_month").sum()["is_canceled"] /  
        ↳hbd["arrival_date_month"].value_counts()).plot.bar(figsize=(16,4))  
plt.ylabel("Cancelations [%]")
```

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



```
[536]: (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")
```

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



- Per quanto riguarda mese e giorno della settimana sono presenti piccole fluttuazioni
- 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

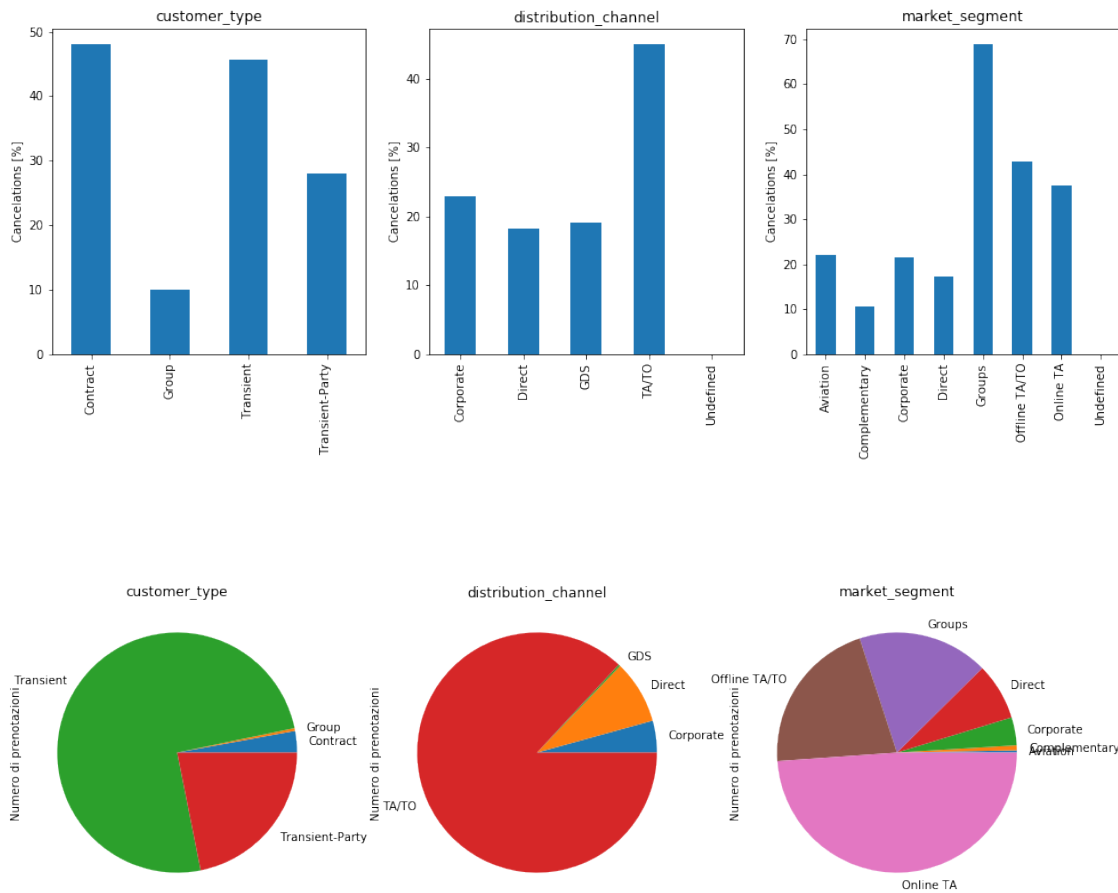
```
[537]: fig = plt.figure(figsize=(16, 5))
ax1 = fig.add_subplot(131)
ax2 = fig.add_subplot(132)
ax3 = fig.add_subplot(133)
ax1.set_ylabel("Cancellations [%]")
ax2.set_ylabel("Cancellations [%]")
ax3.set_ylabel("Cancellations [%]")
ax1.set_title('customer_type')
ax2.set_title('distribution_channel')
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")

```

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



```

[538]: 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 [%]")

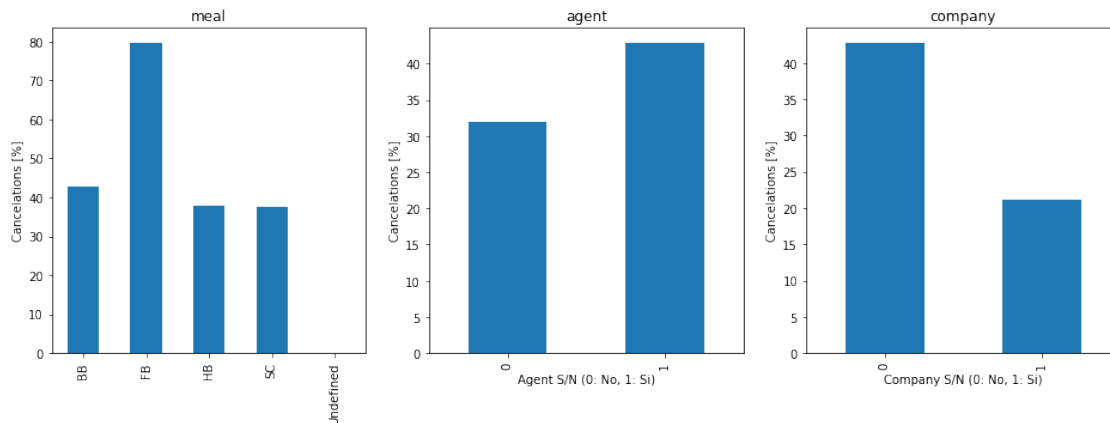
```

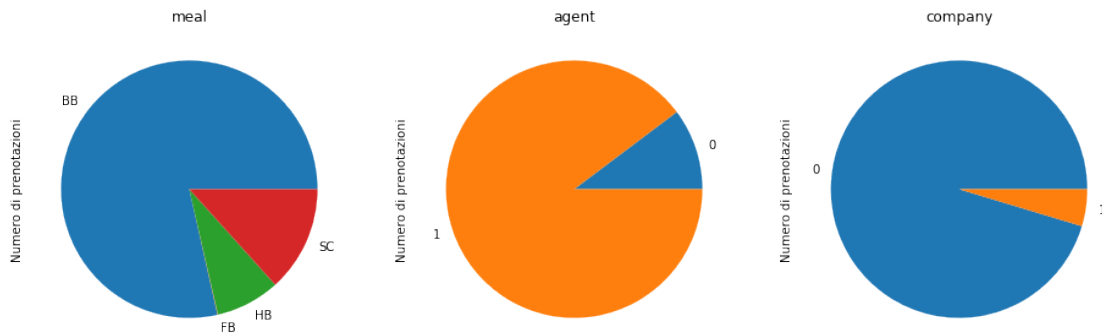
```

(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"

```

[538]: [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

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

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

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

```
[540]: 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

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

- Otteniamo un numero di variabili pari a ...

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

[542]: 248

- Vengono mostrate per completezza tutte le variabili ottenute

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

```
[543]: ['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/T0',
'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

```
[544]: 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 $C=1$

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

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

- Addestriamo il modello sui dati

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

Wall time: 18 s

```
C:\Users\alexandr.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)
```

```
[546]: 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

```
[547]: model.classes_
```

```
[547]: 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...

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

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

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

```
[549]: 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

```
[550]: 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

```
[551]: 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*

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

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

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

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

```
[554]: 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

```
[555]: 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\aleissandr.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 Regularizzazione

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

```
[556]: 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: 24.6 s

C:\Users\aleissandr.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)

```
[556]: 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)
```

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

```
[557]: '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 iniziale *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

```
[558]: 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: 25.2 s

```
C:\Users\aleissandr.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)
```

```
[558]: 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

```
[559]: 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))
```

```
[559]: 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/T0        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

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

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

- Verifichiamo quanti coefficienti hanno valore diverso da zero

```
[561]: 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`

```
[562]: 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_No Deposit ↪ Refund', 'deposit_type_Refundable'], inplace=True)
X_val_no_deposit.drop(columns=['deposit_type_No Deposit', 'deposit_type_No Deposit ↪ 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

```
[563]: 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\aleissandr.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)

```
[563]: 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

```
[564]: 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

```
[565]: 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...

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

```
[566]: ((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

```
[567]: 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:

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

```
[568]: ((52867, 35), (26434, 35))
```

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

```
[569]: 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\aleessandr.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)

```
[569]: 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...

```
[570]: 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

```
[571]: 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 `StratifiedKFold` 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

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

```
[573]: 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

```
[574]: 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)

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

- Effettuiamo quindi la ricerca sui dati

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

```
C:\Users\aleissandr.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\aleissandr.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\aleissandr.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)
```



```

('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

```
[577]: gs.best_params_
```

```
[577]: {'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

```
[578]: 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)
```

```
[578]:   mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
0         2.952332      0.040002         0.043667         0.001248
```

1	3.484012	0.083701	0.044320	0.000466
2	3.386000	0.036120	0.044333	0.000471
3	3.497666	0.034567	0.044667	0.000471
4	3.422000	0.051981	0.044000	0.000815

	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

```
[579]: print_model_information(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

```
[580]: 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: 16.6 s

```
[580]: 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...

```

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

```

[581]: 11

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

```

[582]: 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

```

[583]: 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\aleissandr.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\aleissandr.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\aleissandr.lombardin3\Anaconda3\lib\site-
packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: The max_iter was

```



```

                                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

```

[584]: 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

```

```

[584]:   mean_fit_time  std_fit_time  mean_score_time  std_score_time \
0      23.215530    1.044509      0.117800      0.006794
1       6.379011    0.057216      0.043595      0.001019

   param_poly__degree  params  split0_test_score \
0                   3  {'poly__degree': 3}      0.804899
1                   2  {'poly__degree': 2}      0.804899

   split1_test_score  split2_test_score  split3_test_score  split4_test_score \
0          0.805845          0.801192          0.806583          0.805164
1          0.806601          0.800719          0.805259          0.806015

   mean_test_score  std_test_score  rank_test_score
0          0.804736          0.001866              1
1          0.804699          0.002076              2

```

- Prendiamo il miglior modello e visualizziamone le metriche

```

[585]: 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

```
[586]: 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 $1-\alpha$
- Poiché a noi interessa valutare i modelli con una confidenza del 95%, possiamo ricavare dalle apposite tabelle di valori che, per $1-\alpha = 0.95$ ($\alpha=0.05$), $Z = 1.96$

```
[587]: 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

```
[588]: 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 accurattezze `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

```
[589]: 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

```
[590]: 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

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

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

- Secondo in classifica

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

```
[592]: {'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

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

```
[593]: {'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

```
[594]: 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

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

Wall time: 4.03 s

C:\Users\aleissandr.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)

```
[595]: 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)
```

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

Wall time: 4.6 s

```
C:\Users\aleessandr.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)
```

```
[596]: 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)
```

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

Wall time: 4.65 s

```
C:\Users\aleessandr.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)
```

```
[597]: 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

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

```
[598]: (0.8021507272563478, 0.8116664039673401)
```

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

```
[599]: (0.802227085685336, 0.8117413438206353)
```

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

```
[600]: (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

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

```
[601]: (-0.006652630965746836, 0.006803951235096778)
```

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

```
[602]: (-0.006727789408089268, 0.006727789408089268)
```

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

```
[603]: (-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 volte 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

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

- Addestriamo il modello su tutte le variabili

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

```
C:\Users\aleessandr.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: 24.2 s
```

```
C:\Users\aleessandr.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)
```

```
[605]: 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.

```
[606]: 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 accurattee 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`

```
[607]: 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

```
[608]: 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

```
[609]: 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)

```
[610]: 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)

```
[611]: 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)

```
[612]: 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)

```
[613]: 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)

```
[614]: evaluate_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)

```
[615]: evaluate_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)

```
[616]: evaluate_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

```
[617]: 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_))
```

```
[618]: 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

```
[619]: 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
```

```
[620]: 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
```

```
[621]: 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

```
[622]: 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

```
[623]: 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

```
[624]: 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*

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

- Riporto la configurazione del primo modello

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

- Lo addesto sul dataset...

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

```
C:\Users\aleessandr.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: 7.84 s

```
C:\Users\aleessandr.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)
```

```
[627]: 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

```
[628]: print_model_information(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*

```
[629]: evaluate_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

```
[630]: model_1
```

```
[630]: 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

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

```
[631]: 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)

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

```
[632]: 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/T0       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/T0    -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 considerazione 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 diminuisce...
 - 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 prenotazioni 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 interpretato nel medesimo modo

1.7 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` specifichiamo il tipo di funzione di attivazione
- con `batch_size` specifichiamo la dimensione dei minibatch
- Addestriamo il modello su tutte le istanze dell'hotel *City Hotel*, con solo parte delle variabili ottenute dalla binarizzazione
 - Riutilizziamo `X_train_v2`, `y_train_v2` come training set e `X_val_v2`, `y_val_v2` come validation set
- Definiamo la grid search, specificando il modello, la griglia e lo splitter per la cross-validation (usiamo lo `StratifiedKFold` creato sopra)
 - Per prima cosa realizziamo un modello con sole funzioni di attivazione *identity*

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

```
[646]: model = Pipeline([
        ("scaler", StandardScaler()),
        ("mlp", MLPClassifier(activation="identity", random_state=42))
    ])
    grid = {
        "mlp__hidden_layer_sizes": [16, 32, (16, 8)],
        "mlp__batch_size": [50, 100]
    }
    model_linear = GridSearchCV(model, grid, cv=skf)
    %time model_linear.fit(X_train_v2, y_train_v2);
```

Wall time: 2min 7s

```
[646]: GridSearchCV(cv=StratifiedKFold(n_splits=3, random_state=42, shuffle=True),
                    error_score=nan,
                    estimator=Pipeline(memory=None,
                                       steps=[('scaler',
                                              StandardScaler(copy=True,
                                                             with_mean=True,
                                                             with_std=True)),
                                              ('mlp',
                                              MLPClassifier(activation='identity',
                                                             alpha=0.0001,
                                                             batch_size='auto',
                                                             beta_1=0.9, beta_2=0.999,
                                                             early_stopping=False,
                                                             epsilon=1e-08,
                                                             hidden_layer_sizes=(100,...
                                                             nesterovs_momentum=True,
                                                             power_t=0.5,
                                                             random_state=42,
                                                             shuffle=True,
                                                             solver='adam', tol=0.0001,
```

```

validation_fraction=0.1,
verbose=False,
warm_start=False))],
        verbose=False),
iid='deprecated', n_jobs=None,
param_grid={'mlp__batch_size': [50, 100],
            'mlp__hidden_layer_sizes': [16, 32, (16, 8)]},
pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
scoring=None, verbose=0)

```

- Visualizziamo tutte le parametrizzazioni

```

[647]: result_linear = pd.DataFrame(model_linear.cv_results_)
result_linear.sort_values(by="rank_test_score", inplace=True)
result_linear.reset_index(inplace=True, drop=True)
result_linear

```

```

[647]:
   mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
0      7.893619    1.323134      0.043332    0.002624
1      6.465342    0.579789      0.044666    0.000942
2     10.743837    1.752527      0.044667    0.000472
3      6.809000    0.493364      0.043000    0.000815
4      2.847667    0.304255      0.042333    0.001699
5      3.785334    0.325562      0.042332    0.001247

   param_mlp__batch_size  param_mlp__hidden_layer_sizes  \
0                      50                             16
1                     100                             32
2                      50                             32
3                      50                        (16, 8)
4                     100                             16
5                     100                        (16, 8)

           params  split0_test_score  \
0  {'mlp__batch_size': 50, 'mlp__hidden_layer_siz...    0.806446
1  {'mlp__batch_size': 100, 'mlp__hidden_layer_si...    0.805368
2  {'mlp__batch_size': 50, 'mlp__hidden_layer_siz...    0.806162
3  {'mlp__batch_size': 50, 'mlp__hidden_layer_siz...    0.805538
4  {'mlp__batch_size': 100, 'mlp__hidden_layer_si...    0.804347
5  {'mlp__batch_size': 100, 'mlp__hidden_layer_si...    0.800601

   split1_test_score  split2_test_score  mean_test_score  std_test_score  \
0      0.802974      0.804789      0.804736      0.001418
1      0.804109      0.802974      0.804150      0.000978
2      0.803598      0.802576      0.804112      0.001509
3      0.800817      0.803428      0.803261      0.001931
4      0.802406      0.799852      0.802202      0.001840

```

5	0.801101	0.802520	0.801407	0.000812
---	----------	----------	----------	----------

rank_test_score	
0	1
1	2
2	3
3	4
4	5
5	6

- Andiamo a visualizzare le misure del miglior modello ottenuto

```
[648]: print_model_informations(model_linear, X_train_v2, y_train_v2, X_val_v2,
    ↪ y_val_v2)
```

```
Accuracy = 0.8068018461072861      ( Accuracy on training set =
0.8043770215824616 )
```

```
Intervallo di condidenza = (0.8019980112397259, 0.8115165234193952)
```

```
Precision (Y) = 0.8315508021390374
```

```
Precision (N) = 0.7940772138847519
```

```
Precision = 0.8128140080118946
```

```
Recall (Y) = 0.6749254001265937
```

```
Recall (N) = 0.9016585365853659
```

```
Recall = 0.7882919683559798
```

```
F1 Score (Y) = 0.7450960818567507
```

```
F1 Score (N) = 0.8444552736576006
```

```
F1 Score = 0.7947756777571757
```

```
Matrice di confusione:
```

	N	Y
N	13863	1512
Y	3595	7464

- Mostriamo le misure del nostro miglior modello

```
[649]: 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

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

```
[650]: evaluate_complete_model(model_linear, X_val_v2, y_val_v2, model_1, X_val_v2, y_val_v2)
```

Accuratezza del primo modello: 0.8068018461072861

Accuratezza del secondo modello: 0.8069531663766362

Intervallo di confidenza: (-0.006578475312828371, 0.006881115851528476)

- 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 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 utilizzato precedentemente
- Definiamo la grid search ed effettuiamo la ricerca sui dati

```
[651]: model = Pipeline([
    ("scaler", StandardScaler()),
    ("mlp", MLPClassifier(activation="relu", random_state=42, batch_size=50))
])
grid = {
    "mlp_hidden_layer_sizes": [8, 16, (16, 8)]
}
model_not_linear = GridSearchCV(model, grid, cv=skf)
```

```
%time model_not_linear.fit(X_train_v2, y_train_v2);
```

Wall time: 6min 22s

```
[651]: GridSearchCV(cv=StratifiedKFold(n_splits=3, random_state=42, shuffle=True),
                  error_score=nan,
                  estimator=Pipeline(memory=None,
                                     steps=[('scaler',
                                             StandardScaler(copy=True,
                                                             with_mean=True,
                                                             with_std=True)),
                                             ('mlp',
                                              MLPClassifier(activation='relu',
                                                            alpha=0.0001,
                                                            batch_size=50, beta_1=0.9,
                                                            beta_2=0.999,
                                                            early_stopping=False,
                                                            epsilon=1e-08,
                                                            hidden_layer_sizes=(100,),
                                                            learni...
                                                            momentum=0.9,
                                                            n_iter_no_change=10,
                                                            nesterovs_momentum=True,
                                                            power_t=0.5,
                                                            random_state=42,
                                                            shuffle=True,
                                                            solver='adam', tol=0.0001,
                                                            validation_fraction=0.1,
                                                            verbose=False,
                                                            warm_start=False))],
                  verbose=False),
                  iid='deprecated', n_jobs=None,
                  param_grid={'mlp_hidden_layer_sizes': [8, 16, (16, 8)]},
                  pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                  scoring=None, verbose=0)
```

- Visualizziamo tutte le parametrizzazioni

```
[652]: result_not_linear = pd.DataFrame(model_not_linear.cv_results_)
result_not_linear.sort_values(by="rank_test_score", inplace=True)
result_not_linear.reset_index(inplace=True, drop=True)
result_not_linear
```

```
[652]:   mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
0      48.804324    8.301145      0.050000      0.001414
1      37.599845    6.197406      0.048642      0.001248
2      21.470619    3.048291      0.045006      0.001419
```

	param_mlp__hidden_layer_sizes	params \
0	(16, 8)	{'mlp__hidden_layer_sizes': (16, 8)}
1	16	{'mlp__hidden_layer_sizes': 16}
2	8	{'mlp__hidden_layer_sizes': 8}

	split0_test_score	split1_test_score	split2_test_score	mean_test_score \
0	0.833513	0.831177	0.832879	0.832523
1	0.832094	0.822892	0.832369	0.829118
2	0.827895	0.824764	0.826353	0.826338

	std_test_score	rank_test_score
0	0.000986	1
1	0.004404	2
2	0.001278	3

- Visualizziamo le misure del miglior modello ottenuto

```
[653]: print_model_information(model_not_linear, X_train_v2, y_train_v2, X_val_v2,
    ↪ y_val_v2)
```

```
Accuracy = 0.8381251418627526      ( Accuracy on training set =
0.843021166323037 )
```

```
Intervallo di condidenza = (0.8336357781963568, 0.8425162453280852)
```

```
Precision (Y) = 0.8445121951219512
Precision (N) = 0.8343377124261782
Precision = 0.8394249537740647
```

```
Recall (Y) = 0.7514241794013925
Recall (N) = 0.9004878048780488
Recall = 0.8259559921397206
```

```
F1 Score (Y) = 0.7952533614048518
F1 Score (N) = 0.8661515843473365
F1 Score = 0.8307024728760941
```

```
Matrice di confusione:
```

	N	Y
N	13845	1530
Y	2749	8310

- Notiamo subito una notevole differenza rispetto alle metriche del nostro miglior modello
 - Possiamo notare un importante miglioramento per quanto riguarda la Recall (Y)
- Applichiamo il confronto sull'accuratezza

```
[654]: valuate_complete_model(model_not_linear, X_val_v2, y_val_v2, model_1, X_val_v2,
    ↪ y_val_v2)
```

Accuratezza del primo modello: 0.8381251418627526

Accuratezza del secondo modello: 0.8069531663766362

Intervallo di confidenza: (0.024663945745248723, 0.03768000522698405)

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

```
[655]: model_not_linear.best_estimator_.named_steps["mlp"].coefs_
```

```
[655]: [array([[ 3.47703931e-01,  7.99368080e-01,  8.28971559e-01,
           1.93940687e-01, -5.60852022e-01, -2.32313501e-01,
           4.68681970e-01, -1.25124977e-01,  4.55008715e-01,
           3.06826827e-01, -2.43563351e+00,  1.78653233e-01,
           1.01343666e+00, -2.23960683e-01,  3.83916719e-01,
          -8.10753684e-01],
        [ 1.37369552e-01,  1.77845440e-01, -1.58468356e-02,
           4.74905818e-02,  2.19817557e-01,  8.40890395e-04,
          -2.80856658e-01, -2.22743040e-01, -1.00790329e-01,
          -5.04034414e-02, -2.21123330e-02, -3.20330678e-02,
           3.17298685e-01, -2.16768707e-01, -1.80631831e-01,
           9.60112406e-02],
        [-5.07308880e-02,  4.69122357e-02, -1.18528284e-01,
           9.20332265e-02, -2.25725844e-01,  1.70174573e-01,
           7.15236535e-02, -8.83401384e-02, -3.03053943e-01,
           5.71187527e-02, -2.72903104e-02,  5.32160226e-01,
           2.06817635e-01,  2.56226766e-02, -2.95544116e-01,
           6.48752428e-01],
        [-4.77169829e-02, -4.80352548e-01,  1.42631882e-01,
          -3.92692206e-01,  2.00201637e-01,  3.14457073e-01,
          -1.79914834e-01,  5.30978615e-01,  1.57942883e-01,
          -7.15619532e-01, -6.01132981e-02,  9.80201143e-02,
          -2.42966984e-01,  6.36159892e-02,  2.97171743e-01,
           1.46169082e-01],
        [-1.00350867e+00, -4.99888162e-01, -1.63216450e+00,
           4.57773700e-01, -1.40190823e+00,  9.64783990e-01,
           8.34291510e-01, -1.41333801e+00, -7.75308256e-01,
           5.64770705e-01,  3.33790357e-01,  7.63227821e-01,
           7.07193831e-01, -9.53441918e-01, -4.66517285e-02,
          -1.24816625e+00],
        [ 1.68489887e-01, -1.36430678e+00,  3.33734911e-01,
           1.63351990e-02,  2.88753162e-01, -5.89056273e-01,
           1.57833731e-01,  5.58939980e-01,  1.30077402e+00,
```

-2.57054172e-02, 1.47208880e-01, -3.10590944e-01,
 -1.39239720e-01, 2.27388106e-01, 7.91551605e-01,
 4.81875937e-01],
 [-6.92502860e-02, -1.14913267e+00, -2.99005172e-01,
 2.76921898e-01, -3.80643372e-01, -2.81778701e-01,
 3.32251563e-02, 1.34854484e-01, 8.60727270e-01,
 2.84692755e-01, 5.34766601e-02, -2.48717548e-02,
 -5.20355546e-01, -1.54193803e-01, -7.78711560e-01,
 -2.30173773e-01],
 [3.15288501e-01, -1.25729167e-01, 3.68226162e-02,
 2.42102902e-01, 2.86503558e-03, -2.30497849e-01,
 3.21505544e-02, -8.33232612e-01, 6.42431776e-02,
 -1.41915777e-01, -4.10207314e-01, -6.62433188e-01,
 1.42344353e-01, 2.76463938e-01, 8.20618264e-02,
 2.87069738e-01],
 [-2.20878365e-01, 1.51726448e-01, 3.29398494e-02,
 -6.43064937e-01, 4.07683933e-02, -3.17253543e-01,
 -4.85762963e-01, -4.99030877e-02, -2.39631325e-02,
 9.67378626e-01, -7.35434065e-03, 5.45895198e-01,
 5.91776961e-01, -3.10886608e-01, 3.81293587e-03,
 3.79779400e-02],
 [2.68046436e-01, -2.01459459e-01, -1.40533308e-01,
 5.05096702e-01, -6.03648865e-01, -4.23849209e-01,
 5.35328430e-01, 8.35017161e-02, -3.14113879e-01,
 9.26992677e-01, 1.28281130e+00, -8.03804448e-01,
 5.12430785e-01, 5.75146404e-01, -1.44372982e-01,
 4.61720062e-01],
 [2.88919753e-01, -1.63221915e-01, 4.05534417e-01,
 1.13181125e-01, 5.78127993e-02, -7.90816352e-02,
 -7.32927109e-01, 4.72630027e-04, 3.93116251e-01,
 -9.03680731e-02, 8.82139327e-02, -1.15206102e+00,
 -2.60453134e-01, 1.12941219e-01, -2.75326101e-01,
 4.44108446e-01],
 [-3.69385219e-04, 4.72530896e-02, -4.27951028e-02,
 5.83472578e-04, 1.96462025e-01, 1.80995839e-01,
 -2.51664646e-01, 1.93097929e-01, -5.19585872e-02,
 -1.51313234e-01, -1.17004981e-02, -1.09845694e-03,
 7.20683772e-02, 6.81075495e-02, 3.21268970e-01,
 1.02885349e-01],
 [2.95066925e-01, 3.18608572e-02, 8.54454249e-02,
 1.62239708e-01, 1.23217081e-02, 9.68915059e-02,
 1.16738175e-01, 1.06008559e-01, 2.07183615e-01,
 -1.74399566e-02, -3.08607677e-02, 1.74239414e-01,
 -6.26574967e-02, -2.87255427e-01, 9.60039038e-02,
 -6.13605644e-02],
 [-2.20512483e-01, -1.29563716e-01, -6.36429116e-02,
 -2.78711004e-01, 1.88611931e-01, -6.16111504e-01,

-1.23788222e-01, -3.08084985e-01, -5.41493153e-01,
 3.03533100e-01, -9.44309382e-03, 1.11098819e-01,
 -5.80382535e-02, 3.43852883e-03, -1.18967824e-01,
 -3.83315331e-02],
 [-1.52038111e-01, 2.79336926e-02, 4.00974556e-01,
 -2.53630923e-01, -1.36401347e-01, -7.27401626e-01,
 1.04715758e+00, 1.37528817e-01, 3.89046875e-02,
 -5.69238704e-02, 2.91350304e-02, 2.11957374e-01,
 -1.88089328e-01, -8.25665255e-02, 1.38151902e-01,
 -4.78750714e-01],
 [4.81672131e-02, 2.28344332e-01, 1.25496143e-01,
 -6.65792710e-02, -4.69638562e-01, 4.62367491e-01,
 -4.25770130e-01, 7.27851343e-02, 1.67952645e-01,
 1.92805456e-01, 1.80236295e-01, -5.24984473e-01,
 1.06957275e-01, -1.55924372e-01, -6.57341822e-02,
 -9.45316240e-02],
 [2.38910382e-01, 1.16815028e-01, -1.90188382e-01,
 -1.21576135e-01, -4.19442693e-02, -3.38511082e-01,
 -6.04873110e-01, -2.38260259e-01, 3.59591803e-01,
 1.05464290e-01, 9.81763222e-03, -5.99812871e-02,
 1.26832119e-02, -8.76999381e-03, -1.85304417e-02,
 1.23272934e-01],
 [3.65620079e-01, 4.06657469e-01, 1.19722987e-01,
 3.50075313e-02, 1.29040838e-01, -3.59034569e-01,
 2.57650697e-01, -2.88363237e-02, 2.69523874e-01,
 -6.47438272e-01, 4.84943045e-02, -3.53888747e-01,
 -1.56735711e-01, -5.85976836e-01, 7.80846514e-03,
 1.15767000e-01],
 [-1.16660636e-01, 1.63606497e-02, -6.39197109e-01,
 -1.11487440e-01, 2.84918669e-02, -1.68375739e-01,
 -6.87415298e-01, -1.49650429e-01, -7.45174045e-01,
 1.39205214e-01, -1.24784515e-02, 2.96485778e-04,
 -1.97270581e-01, 4.58374476e-02, -1.18417597e-02,
 -4.73132685e-01],
 [1.26444611e-01, 1.66778904e-01, 3.24596009e-01,
 -2.87061952e-02, 2.00300383e-01, -1.09688424e-01,
 -1.54234202e-01, -3.43636352e-01, -8.27296719e-01,
 4.33570613e-01, 4.85960850e-03, 7.60944288e-02,
 3.63340109e-01, -2.77878464e-02, -5.00908667e-01,
 -5.98453430e-01],
 [8.31135763e-02, 1.68921313e-01, 2.47768421e-02,
 -2.27876039e-01, 3.55434332e-02, -6.10718843e-03,
 -1.96029618e-01, -8.54849428e-02, -4.88001320e-01,
 1.09007198e-02, 1.90083147e-02, 4.24595399e-02,
 -4.99336584e-01, 8.03504002e-03, -4.37434958e-01,
 -4.98486190e-01],
 [3.07827622e-01, 1.19288118e-01, -1.95995596e-01,

-9.56872809e-02, -5.34828337e-02, -4.36120463e-01,
 -2.54726815e-01, -4.45901517e-01, 2.73843352e-01,
 2.51458405e-01, -4.64541662e-02, -2.39832308e-01,
 1.38652279e-01, 2.31390620e-01, -5.40349692e-02,
 -1.10518732e-01],
 [5.60201307e-02, -1.07014544e-01, -2.67126005e-01,
 4.33248247e-01, -8.94970494e-02, -3.47225616e-01,
 3.24680814e-01, -1.17081727e-01, 1.09079646e-01,
 -2.97280819e-02, 3.15114287e-02, 2.89014727e-02,
 1.36129177e-01, -3.63087279e-01, 4.63514201e-02,
 3.10358758e-01],
 [-1.66766220e-01, -9.95692275e-02, -9.15005916e-02,
 2.62747364e-02, -3.99278374e-01, 3.01570832e-01,
 -4.63930817e-01, -1.76733794e-01, 9.53661221e-02,
 7.53557561e-02, 1.20275753e-01, -5.36784401e-01,
 -2.33691087e-01, -1.28901479e-01, 1.92263382e-01,
 1.94244094e-01],
 [2.44323578e-01, -1.90226370e-01, -1.89719814e-01,
 -5.30003666e-01, 1.57724864e-01, -3.57311015e-02,
 -8.68166720e-02, -3.65970325e-01, -4.07249339e-01,
 1.24261597e-01, 3.29812142e-02, 1.54101122e-01,
 5.99575636e-02, -1.67516265e-01, 5.61509079e-02,
 2.85599276e-01],
 [-6.35951245e-01, 3.80820372e-01, -1.05368301e-02,
 4.77320862e-01, -6.45443612e-01, 7.45452055e-02,
 -5.66794160e-01, -1.21035180e-01, 6.69955634e-01,
 -3.54282707e-02, 7.40942772e-02, 1.61602933e-01,
 5.90171580e-01, 4.02670134e-02, -5.45510812e-01,
 7.36585943e-02],
 [-1.22783175e+00, 2.11691988e-01, -1.09969289e-01,
 3.01324137e-01, -1.58106740e-01, 1.07550523e+00,
 -6.97371186e-01, 4.24488461e-01, -7.01848923e-01,
 5.82069698e-03, 4.90903250e-02, 6.38880541e-01,
 -8.83854947e-01, -1.30204227e+00, -8.71639285e-01,
 3.45320465e-01],
 [4.51876440e-02, -1.15419965e-01, -3.43012936e-01,
 4.42268117e-01, -6.40242800e-01, -3.77480387e-01,
 -1.62113007e-01, -2.42760716e-01, -6.72729242e-01,
 3.41680789e-01, 8.72703727e-03, -6.77627277e-02,
 5.82926938e-02, 1.03866933e-01, 4.20277288e-01,
 -2.16810932e-01],
 [2.62940485e-01, -8.94712582e-01, 6.58169201e-01,
 8.82618219e-01, -8.27001247e-01, 5.37813936e-01,
 5.33225879e-02, -2.02804300e-01, 9.49747184e-02,
 1.82408799e-01, -3.88699748e-02, 3.32899004e-01,
 -6.02822184e-02, 4.42334919e-01, 1.36675705e-01,
 2.45586074e-01],

```

[-8.02022476e-01, -2.10246322e-01, -2.82224259e-01,
 8.69624459e-01, -2.63972649e-01, -2.11325084e-01,
-1.54222336e+00, 6.00176580e-01, -1.14642410e+00,
 5.72284794e-01, -1.96432954e-02, 4.79908232e-01,
-9.02057487e-01, -1.12791513e+00, 5.80744254e-01,
-2.83131905e-01],
[ 6.02608936e-01, 1.68662827e-01, 2.88803945e-01,
 8.78999999e-02, -5.04579653e-01, 3.42392606e-01,
 1.11383098e+00, 6.62679236e-01, 3.24239055e-01,
 9.80612610e-02, 3.58641513e-02, 3.12622803e-01,
 7.13047294e-01, 2.46284513e-01, -5.10850779e-03,
-6.73329018e-01],
[ 1.20594396e-01, -1.09706524e+00, 5.28758793e-01,
 2.38624379e-01, 3.15590926e-01, -4.50847318e-01,
-5.34407597e-01, 3.60263667e-01, 1.82720020e-01,
-2.94744269e-01, -1.20251472e-02, -4.03346889e-01,
 6.12140813e-01, 1.51867794e-01, -4.77617702e-01,
 6.07909137e-01],
[ 2.25821838e-01, 8.58662520e-01, -4.70526429e-01,
-2.18401633e-01, -2.41770393e-01, 6.93439295e-01,
 6.72887256e-01, -3.76855560e-01, -2.71511495e-02,
 2.83148816e-01, 7.75307187e-01, 7.84134229e-01,
-1.61922661e-01, -5.13020146e-02, 3.83445190e-01,
-7.40360860e-01],
[-1.13694748e+00, 9.91657679e-01, 3.95488863e-01,
 1.20499137e+00, 1.27455568e-01, -7.00747315e-01,
 6.45450456e-01, 9.90250373e-02, 4.53464379e-01,
-4.30382428e-01, -6.80768635e-01, -1.00667564e-02,
-1.81540018e-01, -7.03976474e-01, 1.07447254e-01,
-3.27627529e-01],
[ 2.19354686e-01, -5.86994027e-01, -1.88260020e+00,
-5.77615162e-01, -7.25746797e-01, 1.62423152e-02,
-3.96306192e-01, 6.20927103e-01, 1.31479921e-01,
 6.67469817e-01, 4.80105142e-01, 3.17791218e-02,
 2.47456854e-01, -9.18687612e-01, 1.02439726e+00,
 4.05240474e-01]]),
array([[ -1.66560430e-01, 1.00812526e+00, 6.47222512e-01,
        -1.49475090e-01, -3.06792495e-01, 5.98074854e-01,
        -4.00047038e-02, -4.15677161e-01],
       [ -2.35461144e-01, -5.70048050e-01, -1.67101419e-01,
        -6.93286485e-02, -1.54800897e-01, -2.44665883e+00,
        -4.82934703e-01, 2.96900119e-01],
       [ -1.67137894e+00, -3.65075388e-01, 2.26024374e-01,
        -1.16018176e-01, -1.92888419e-01, 1.55989371e-01,
        -2.30275190e-01, -8.73795239e-01],
       [ -3.16815152e-01, 9.89301748e-05, 1.88188608e-01,
        -1.39124581e+00, -1.20291892e+00, 5.13684453e-02,

```

```

2.70755067e-01, -2.03684938e-01],
[ 2.13778234e-01,  3.88975017e-01, -1.83367646e-01,
-2.82127984e-02,  3.80103904e-01, -1.17242812e+00,
 4.55410657e-01,  3.46421383e-01],
[-3.58256139e-01, -2.51212596e-01,  2.03925109e-01,
 5.79981974e-01, -1.11205155e+00, -3.44646085e-02,
-3.55936806e-01,  2.09724697e-01],
[-2.65527680e-01,  5.44631267e-01,  1.10709401e-01,
 2.18034247e-01, -3.98653217e-02, -4.64519016e-01,
 8.48513056e-01,  1.12531831e+00],
[ 1.83815090e-01,  8.11606610e-01,  1.67646950e-01,
-2.21007176e-01,  2.16280338e-01, -1.04373431e-01,
-1.03336876e-01,  1.27371670e-01],
[-6.34812014e-01,  9.07197515e-02, -7.68712876e-01,
-1.84269840e+00, -4.03717495e-01,  1.38030542e-01,
 1.65202736e-01,  1.29217171e-01],
[ 5.13512342e-01,  5.20693009e-01, -5.02243940e-01,
-4.19858753e-01,  5.37075848e-01, -6.58593766e-02,
-1.31967945e-01, -8.02790779e-01],
[ 6.08983645e-01,  9.45688896e-02,  1.93161358e-01,
 2.64659248e-01, -2.47775553e+00,  1.02281235e+00,
 1.00465511e+00, -1.79486097e+00],
[ 1.82276865e-01, -2.49861969e-01, -1.96897505e+00,
-2.15793304e-01,  1.90526458e-01,  2.51315038e-01,
-1.34228332e-01,  3.57483477e-01],
[-1.04051342e+00, -4.55189538e-02,  2.33091254e-02,
 7.59018131e-01, -4.79017497e-01, -1.21318095e-01,
 6.63621493e-01,  2.08683509e-01],
[ 6.84515278e-01, -6.36261751e-02,  3.44676242e-01,
-9.18458987e-02,  2.74098630e-01, -3.23677999e+00,
 5.59359170e-01, -8.07179083e-02],
[ 8.47358023e-02, -3.22222425e-01,  1.45957529e-01,
-5.63165945e-02, -3.47884289e-01, -5.46248824e-03,
-7.62389535e-01,  1.40228442e-01],
[-2.13822987e-01,  1.31235531e-01, -9.20449573e-01,
-5.74521400e-02,  7.42248014e-02, -1.07684915e+00,
-1.90674112e-01, -5.51199516e-01]]),
array([[ -0.74822095],
[ -0.6162039 ],
[ -1.19692408],
[  1.23998588],
[  0.98553381],
[  1.35211232],
[ -0.8004273 ],
[  0.96262429]])]

```

```
[656]: model_not_linear.best_estimator_.named_steps["mlp"].intercepts_
```

```
[656]: [array([ 0.20442423, -0.23905167, -0.27948889, -0.62805007,  0.32014238,  
              0.17386866,  0.7011972 ,  0.18032817,  0.59139784, -0.8459819 ,  
              -0.74398111,  0.81966984,  0.16298867,  0.14003365,  0.01080342,  
              0.47374748]),  
array([ 0.342573 ,  0.94898195, -0.03831515,  0.48540371, -0.14517797,  
        0.65567142,  0.41571985,  0.39216435]),  
array([0.9405924])]
```

- 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 dovrei rinunciare alla possibilità di sapere su cosa le nostre decisioni si basano