BANK MARKETING ANALYSIS

“Marketing is the process by which companies create value for customers and build strong customer relationships to capture value from customers in return.” Kotler and Armstrong (2010).

Il dataset utilizzato per l’analisi è stato importato da Uci Machine Learning Repository. Il dataset contiene dati relative ad una campagna di marketing bancario e lo scopo dell’analisi è quello di trarre insights e raccomandazioni al fine di ottimizzare le future campagne di marketing e, in questo caso specifico, attrarre maggiori clienti ed aumentare il tasso di conversione relativo alla sottoscrizione di un conto deposito a termine . L’approccio, al fine di ottimizzare la campagna di marketing sulla scorta del nostro dataset, comprende i seguenti passaggi: • Importazione del dataset ed analisi esplorativa sulle singole variabili e sulla relazione di ogni variabile con la variabile di risposta (ovvero il risultato della campagna di marketing); • Pulizia e trasformazione dei dati: rimozione di colonne irrilevanti, gestione dei valori mancanti, gestione degli outliers, trasformazione delle colonne categoriche in variabili numeriche (One-Hot-Encoding) e delle variabili categoriche binarie (“yes” e “no”) in colonne che contengano solo valori booleani; • Utilizzare tecniche di Machine Learning per prevedere il risultato della campagna di marketing e per identificare quali sono i fattori che influenzano maggiormente il successo della campagna stessa (Feature Importance). Nello specifico, le tecniche di ML utilizzate sono state le seguenti:

- Logistic Regression
- Decision tree
- Random Forest

Importo le Librerie

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [ ]:
```
import warnings
warnings.filterwarnings("ignore")
```

Importo il dataset

In [ ]:
```
df_raw = pd.read_csv('bank_dataset.csv', sep =';')
```

ANALISI ESPLORATIVA

In [ ]:
```
print(df_raw.head())
print(df_raw.shape)
print(df_raw.info())
```

```
   age           job  marital  education default  balance housing loan  \
0   58    management  married   tertiary      no     2143     yes   no
1   44    technician   single  secondary      no       29     yes   no
2   33  entrepreneur  married  secondary      no        2     yes  yes
3   47   blue-collar  married    unknown      no     1506     yes   no
4   33       unknown   single    unknown      no        1      no   no

   contact  day month  duration  campaign  pdays  previous poutcome   y
0  unknown    5   may       261         1     -1         0  unknown  no
1  unknown    5   may       151         1     -1         0  unknown  no
2  unknown    5   may        76         1     -1         0  unknown  no
3  unknown    5   may        92         1     -1         0  unknown  no
4  unknown    5   may       198         1     -1         0  unknown  no
(45211, 17)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   age        45211 non-null  int64
 1   job        45211 non-null  object
 2   marital    45211 non-null  object
 3   education  45211 non-null  object
 4   default    45211 non-null  object
 5   balance    45211 non-null  int64
 6   housing    45211 non-null  object
 7   loan       45211 non-null  object
 8   contact    45211 non-null  object
 9   day        45211 non-null  int64
 10  month      45211 non-null  object
 11  duration   45211 non-null  int64
 12  campaign   45211 non-null  int64
 13  pdays      45211 non-null  int64
 14  previous   45211 non-null  int64
 15  poutcome   45211 non-null  object
 16  y          45211 non-null  object
dtypes: int64(7), object(10)
memory usage: 5.9+ MB
None
```

45211 righe, 17 colonne 45211 valori non nulli per ogni colonna 10 variabili categoriche e 7 variabili numeriche

Descrizione colonne categoriche

In [ ]:
```
for col in df_raw.select_dtypes(include='object').columns:
    print(col)
    print(df_raw[col].unique())
```

```
job
['management' 'technician' 'entrepreneur' 'blue-collar' 'unknown'
 'retired' 'admin.' 'services' 'self-employed' 'unemployed' 'housemaid'
 'student']
marital
['married' 'single' 'divorced']
education
['tertiary' 'secondary' 'unknown' 'primary']
default
['no' 'yes']
housing
['yes' 'no']
loan
['no' 'yes']
contact
['unknown' 'cellular' 'telephone']
month
['may' 'jun' 'jul' 'aug' 'oct' 'nov' 'dec' 'jan' 'feb' 'mar' 'apr' 'sep']
poutcome
['unknown' 'failure' 'other' 'success']
y
['no' 'yes']
```

Statistica descrittiva di base per le variabili numeriche

In [ ]:
```
df_raw.describe()
```

Out[ ]:

	age	balance	day	duration	campaign	p
count	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.00
mean	40.936210	1362.272058	15.806419	258.163080	2.763841	40.19
std	10.618762	3044.765829	8.322476	257.527812	3.098021	100.12
min	18.000000	-8019.000000	1.000000	0.000000	1.000000	-1.00
25%	33.000000	72.000000	8.000000	103.000000	1.000000	-1.00
50%	39.000000	448.000000	16.000000	180.000000	2.000000	-1.00
75%	48.000000	1428.000000	21.000000	319.000000	3.000000	-1.00
max	95.000000	102127.000000	31.000000	4918.000000	63.000000	871.00

Valori mancanti

In [ ]:
```
missing_values = df_raw.isnull().sum()
print("Missing values: ",missing_values)
```

```
Missing values:  age          0
job          0
marital      0
education    0
default      0
balance      0
housing      0
loan         0
contact      0
day          0
month        0
duration     0
campaign     0
pdays        0
previous     0
poutcome     0
y            0
dtype: int64
```

Non ci sono valori mancanti Rinominiamo il dataset

In [ ]:
```
df_wrgld = df_raw
print(df_wrgld.head)
```

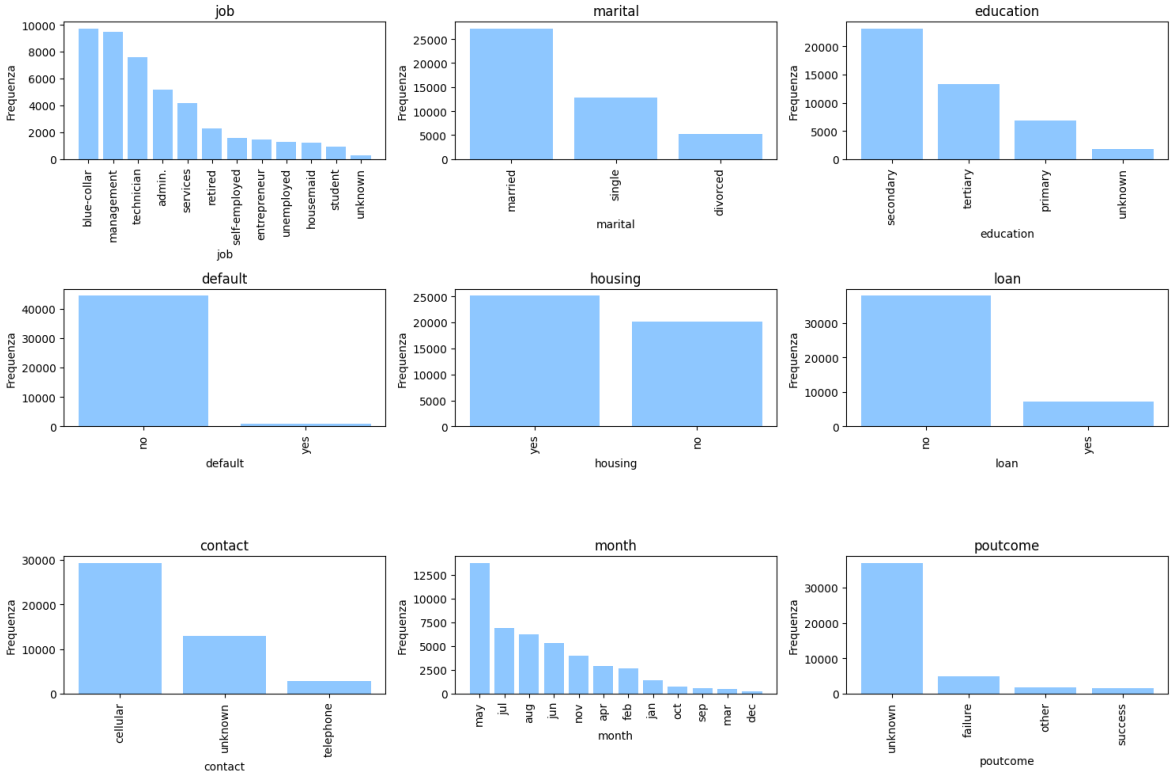
```
<bound method NDFrame.head of        age           job   marital  education defau
lt  balance housing loan  \
0       58    management   married   tertiary      no     2143     yes   no
1       44    technician    single  secondary      no       29     yes   no
2       33  entrepreneur   married  secondary      no        2     yes  yes
3       47   blue-collar   married    unknown      no     1506     yes   no
4       33       unknown    single    unknown      no        1      no   no
...    ...           ...       ...        ...     ...      ...     ...  ...
45206   51    technician   married   tertiary      no      825      no   no
45207   71       retired  divorced    primary      no     1729      no   no
45208   72       retired   married  secondary      no     5715      no   no
45209   57   blue-collar   married  secondary      no      668      no   no
45210   37  entrepreneur   married  secondary      no     2971      no   no

         contact  day month  duration  campaign  pdays  previous poutcome    y
0        unknown    5   may       261         1     -1         0  unknown   no
1        unknown    5   may       151         1     -1         0  unknown   no
2        unknown    5   may        76         1     -1         0  unknown   no
3        unknown    5   may        92         1     -1         0  unknown   no
4        unknown    5   may       198         1     -1         0  unknown   no
...          ...  ...   ...       ...       ...    ...       ...      ...  ...
45206   cellular   17   nov       977         3     -1         0  unknown  yes
45207   cellular   17   nov       456         2     -1         0  unknown  yes
45208   cellular   17   nov      1127         5    184         3  success  yes
45209  telephone   17   nov       508         4     -1         0  unknown   no
45210   cellular   17   nov       361         2    188        11    other   no

[45211 rows x 17 columns]>
```

Esplorazione delle variabili categoriche

In [ ]:
```
print(df_wrgld.select_dtypes(include='object').columns.to_list())
categorical_variables = ['job', 'marital', 'education', 'default','housing', 'lo

num_cols = 3
num_rows = 3
```

```
plt.figure(figsize=(15, 10))

for i, column in enumerate(categorical_variables, start=1):
    plt.subplot(num_rows, num_cols, i)
    values = df_wrgld[column].value_counts()
    plt.bar(values.index, values, color='DodgerBlue', alpha=0.5)
    plt.xticks(rotation=90)
    plt.title(f'{column}')
    plt.xlabel(column)
    plt.ylabel('Frequenza')
plt.tight_layout()
plt.show()
```

```
['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month',
'poutcome', 'y']
```



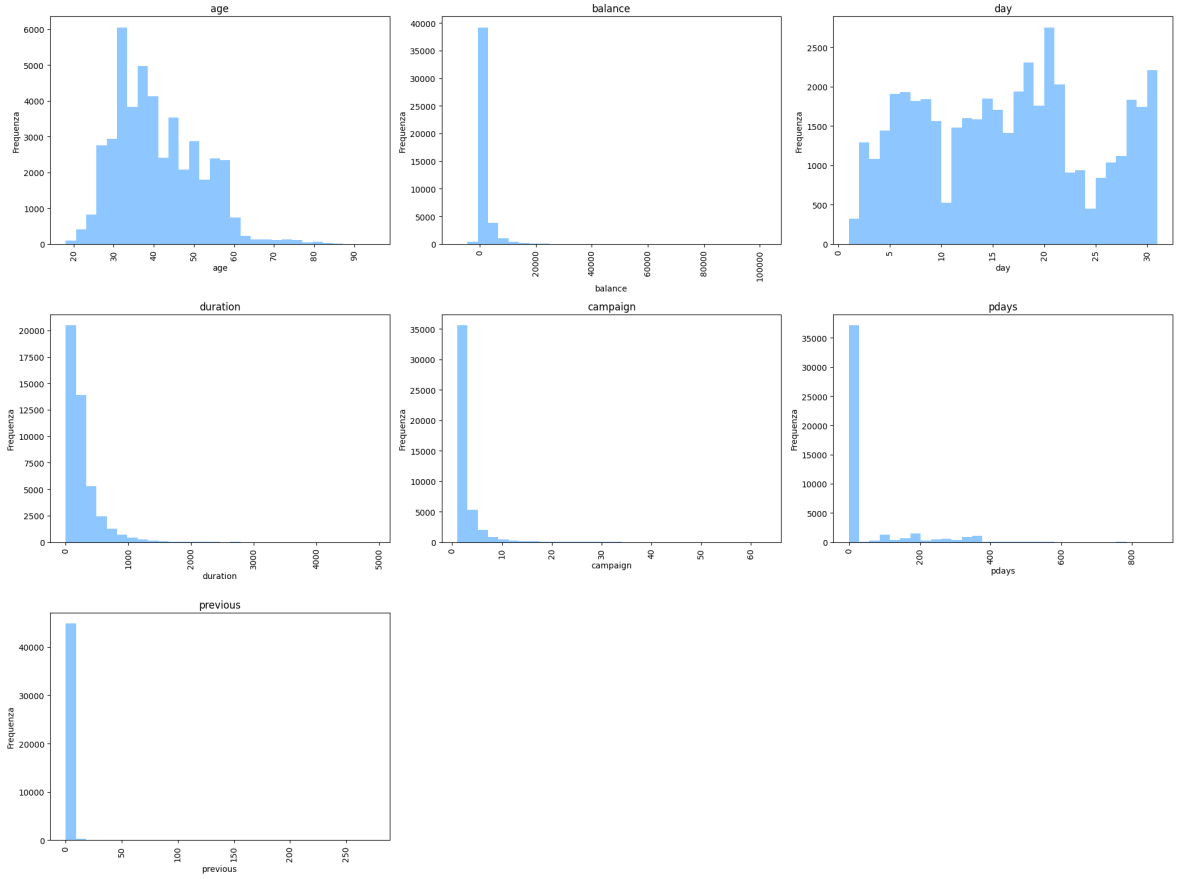
Esplorazione delle variabili categoriche

In [ ]:

```
print(df_wrgld.select_dtypes(include='number').columns.to_list())
numerical_variables = ['age', 'balance', 'day', 'duration', 'campaign', 'pdays',
df_wrgld[numerical_variables].describe()

num_cols = 3
num_rows = 3

plt.figure(figsize=(20, 15))

for i, column in enumerate(numerical_variables, start=1):
    plt.subplot(num_rows, num_cols, i)
    values = df_wrgld[column].value_counts()
    plt.hist(df_wrgld[column], color='DodgerBlue', alpha=0.5, bins=30)
    plt.xticks(rotation=90)
    plt.title(f'{column}')
    plt.xlabel(column)
    plt.ylabel('Frequenza')
```

```
plt.tight_layout()
plt.show()
```

```
['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous']
```



Gestione degli Outlier

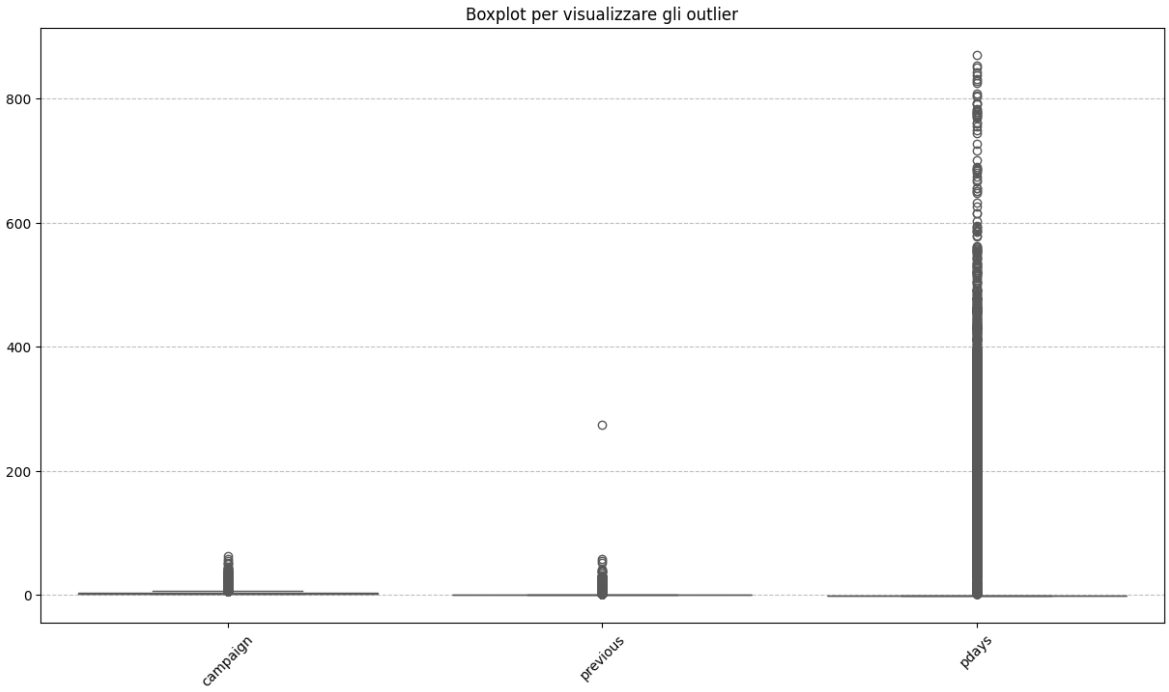
In [ ]:
```
df_wrgld[numerical_variables].describe()
```

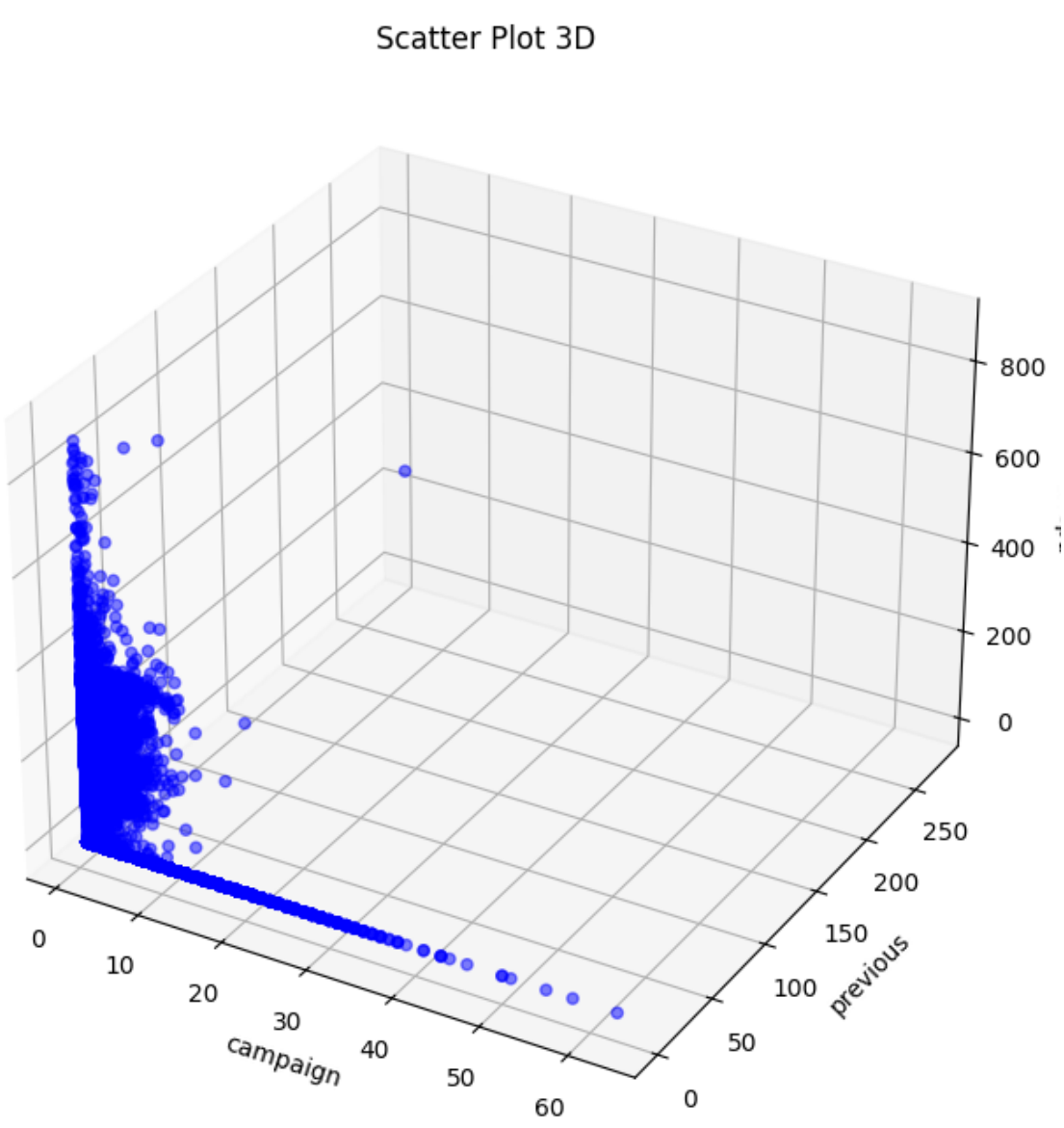
Out[ ]:

	age	balance	day	duration	campaign	p
count	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.00
mean	40.936210	1362.272058	15.806419	258.163080	2.763841	40.19
std	10.618762	3044.765829	8.322476	257.527812	3.098021	100.12
min	18.000000	-8019.000000	1.000000	0.000000	1.000000	-1.00
25%	33.000000	72.000000	8.000000	103.000000	1.000000	-1.00
50%	39.000000	448.000000	16.000000	180.000000	2.000000	-1.00
75%	48.000000	1428.000000	21.000000	319.000000	3.000000	-1.00
max	95.000000	102127.000000	31.000000	4918.000000	63.000000	871.00

Le colonne 'pdays', 'campaign' and 'previous', come evince dal barplot precedent, presentano degli outlier: analizziamo le tre colonne attraverso rappresentazione grafica di Boxplot e Scatterplot

In [ ]:
```
num_otlr_vbls = ['campaign', 'previous', 'pdays']
plt.figure(figsize=(15,8))

sns.boxplot(data=df_wrgld[num_otlr_vbls], palette='Set2')
plt.xticks(rotation=45)
plt.grid(True, axis='y', linestyle='--', alpha=0.8)
plt.title('Boxplot per visualizzare gli outlier')
plt.show()


from mpl_toolkits.mplot3d import Axes3D
x = df_wrgld['campaign']
y = df_wrgld['previous']
z = df_wrgld['pdays']
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x, y, z, c='blue', marker='o', alpha=0.5)
ax.set_xlabel('campaign')
ax.set_ylabel('previous')
ax.set_zlabel('pdays')
ax.set_title('Scatter Plot 3D')
plt.show()
```





La colonna 'pdays' presenta un min=-1, analizziamo quindi la percentuale di questo valore sul totale delle osservazioni

In [ ]:
```
count_minus_1 = (df_wrgld['pdays'] == -1).sum()
tot_values = len(df_wrgld['pdays'])
perc_minus_1 = (count_minus_1 / tot_values) *100
print(count_minus_1)
print(perc_minus_1)
```

```
36954
81.73674548229414
```

"-1" probabilmente sta ad indicare che il cliente non è stato contattato precedentemente oppure indica dei dati mancanti. Dal momento che non siamo certi di questo e che "-1" rappresenta più del 50% (81%) dei valori totali di "pdays", eliminiamo questa colonna

In [ ]:
```
df_wrgld = df_wrgld.drop('pdays', axis=1)
print(df_wrgld.head())
numerical_variables =  ['age', 'balance', 'day', 'duration', 'campaign', 'previo
```

```
   age           job  marital  education default  balance housing loan  \
0   58    management  married   tertiary      no     2143     yes   no
1   44    technician   single  secondary      no       29     yes   no
2   33  entrepreneur  married  secondary      no        2     yes  yes
3   47   blue-collar  married    unknown      no     1506     yes   no
4   33       unknown   single    unknown      no        1      no   no

   contact  day month  duration  campaign  previous poutcome   y
0  unknown    5   may       261         1         0  unknown  no
1  unknown    5   may       151         1         0  unknown  no
2  unknown    5   may        76         1         0  unknown  no
3  unknown    5   may        92         1         0  unknown  no
4  unknown    5   may       198         1         0  unknown  no
```

Controllo la % di valori maggiori di 10 per 'campaign' e 'previous' Le due variabili rappresentano rispettivamente: il numero di contatti eseguiti durante la campagna in analisi e durente la campagna precedente per un determinato cliente.

In [ ]:
```
print(len (df_wrgld[df_wrgld['campaign'] > 10]) / len(df_wrgld) * 100)
print(len (df_wrgld[df_wrgld['previous'] > 10]) / len(df_wrgld) * 100)
```

```
2.645373913428148
0.6502842228661166
```

Dal momento che i valori in questione rappresnetano una % minima del total dei valori e che, verosimilmente, un numero maggiore di 10 contatti per cliente potrebbe non essere commercialmente auspicabile, li sostituisco con la loro rispettiva mediana.

In [ ]:
```
subst_outliers = ['campaign', 'previous']
for var in subst_outliers:
    mean_value = df_wrgld[var].median()
    df_wrgld.loc[df_wrgld[var] > 10, var] = mean_value

df_wrgld[numerical_variables].describe()
```

Out[ ]:

	age	balance	day	duration	campaign	prev
count	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.00
mean	40.936210	1362.272058	15.806419	258.163080	2.371790	0.46
std	10.618762	3044.765829	8.322476	257.527812	1.749001	1.30
min	18.000000	-8019.000000	1.000000	0.000000	1.000000	0.00
25%	33.000000	72.000000	8.000000	103.000000	1.000000	0.00
50%	39.000000	448.000000	16.000000	180.000000	2.000000	0.00
75%	48.000000	1428.000000	21.000000	319.000000	3.000000	0.00
max	95.000000	102127.000000	31.000000	4918.000000	10.000000	10.00

Riguardo la variabile 'duration', la quale rappresenta la durata dell'ultimo contatto: questo attributo influenza fortemente il risultato desiderato (ad esempio, se la durata è 0, allora y='no'); tuttavia, la durata non è nota prima che una chiamata sia effettuata. Inoltre, dopo la fine della chiamata, y è ovviamente noto. Dunque, la decisione è quella di

rimuovere questa variabile poichè impatta molto sul modello predittivo realistico che ho intenzione di implementare.

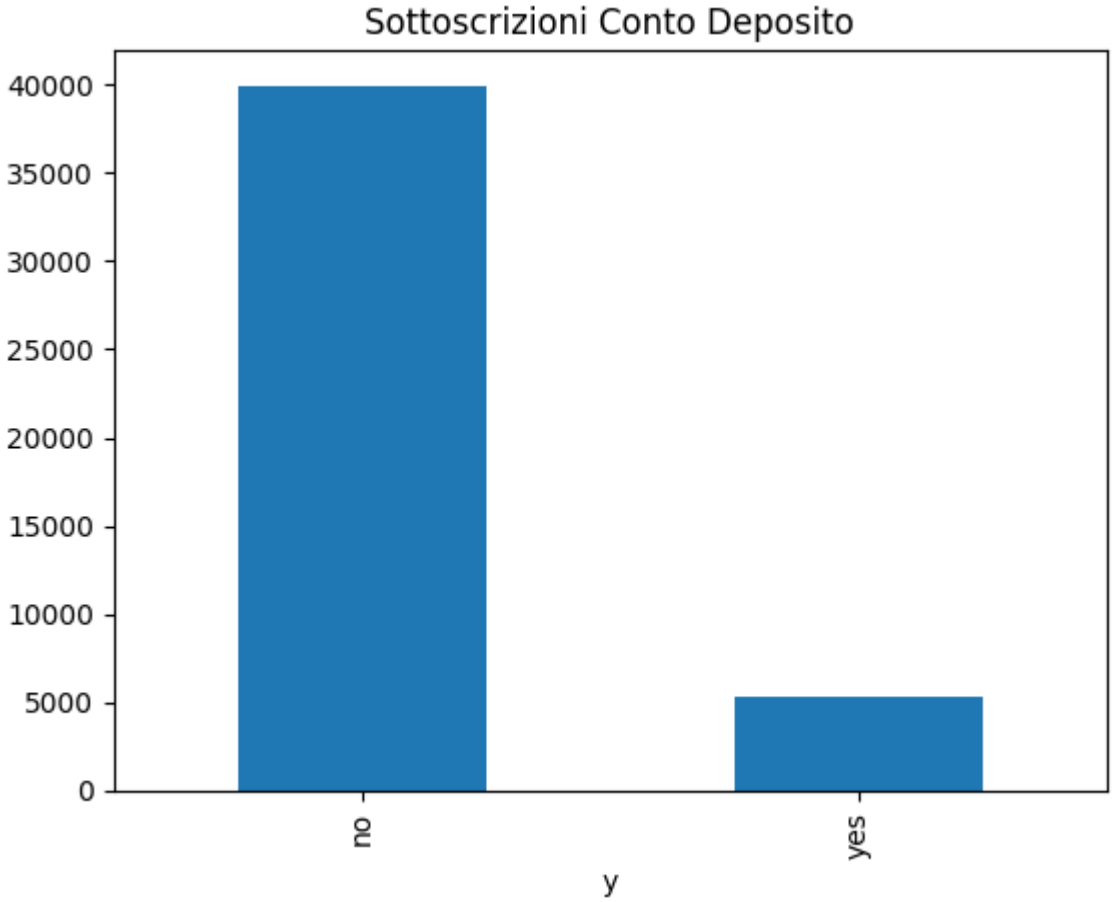
In [ ]:
```
df_wrgld = df_wrgld.drop('duration', axis=1)
print(df_wrgld.head())
```

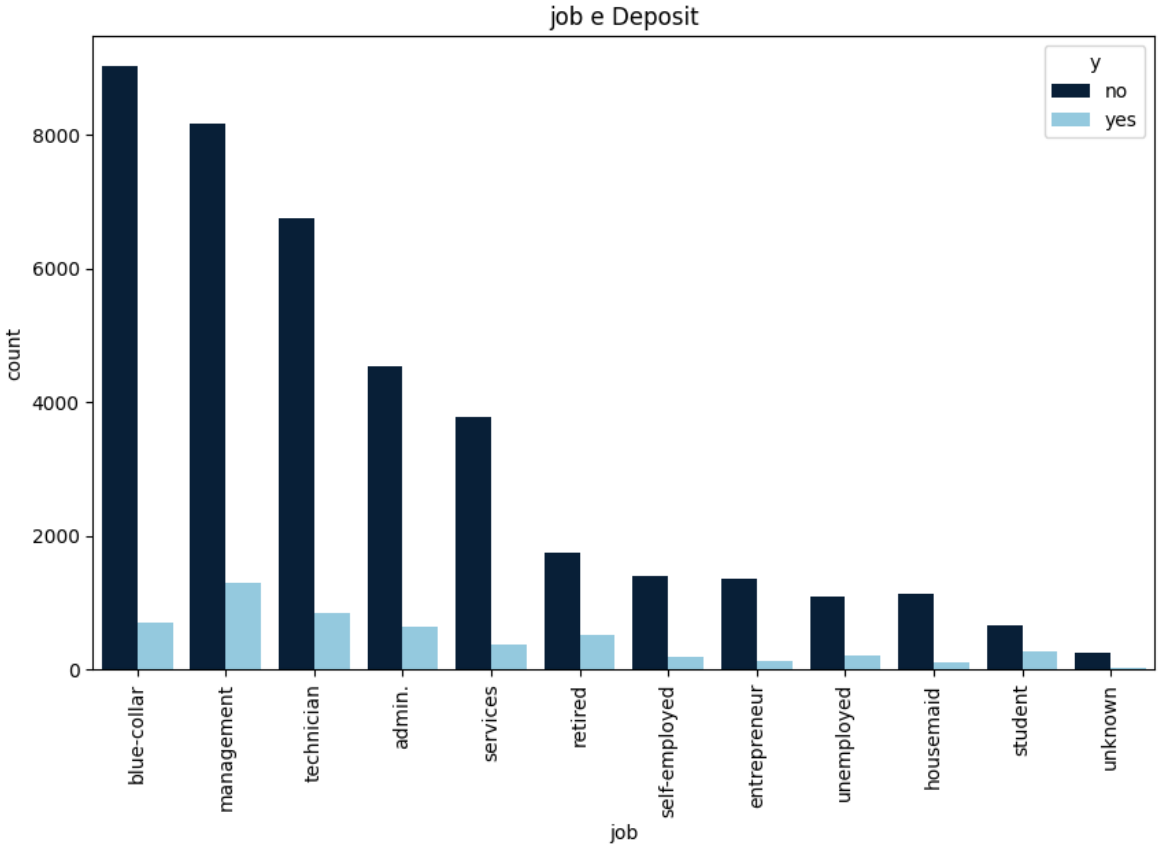
```
   age           job  marital  education default  balance housing loan  \
0   58    management  married   tertiary      no     2143     yes   no
1   44    technician   single  secondary      no       29     yes   no
2   33  entrepreneur  married  secondary      no        2     yes  yes
3   47   blue-collar  married    unknown      no     1506     yes   no
4   33       unknown   single    unknown      no        1      no   no

   contact  day month  campaign  previous poutcome   y
0  unknown    5   may         1         0  unknown  no
1  unknown    5   may         1         0  unknown  no
2  unknown    5   may         1         0  unknown  no
3  unknown    5   may         1         0  unknown  no
4  unknown    5   may         1         0  unknown  no
```

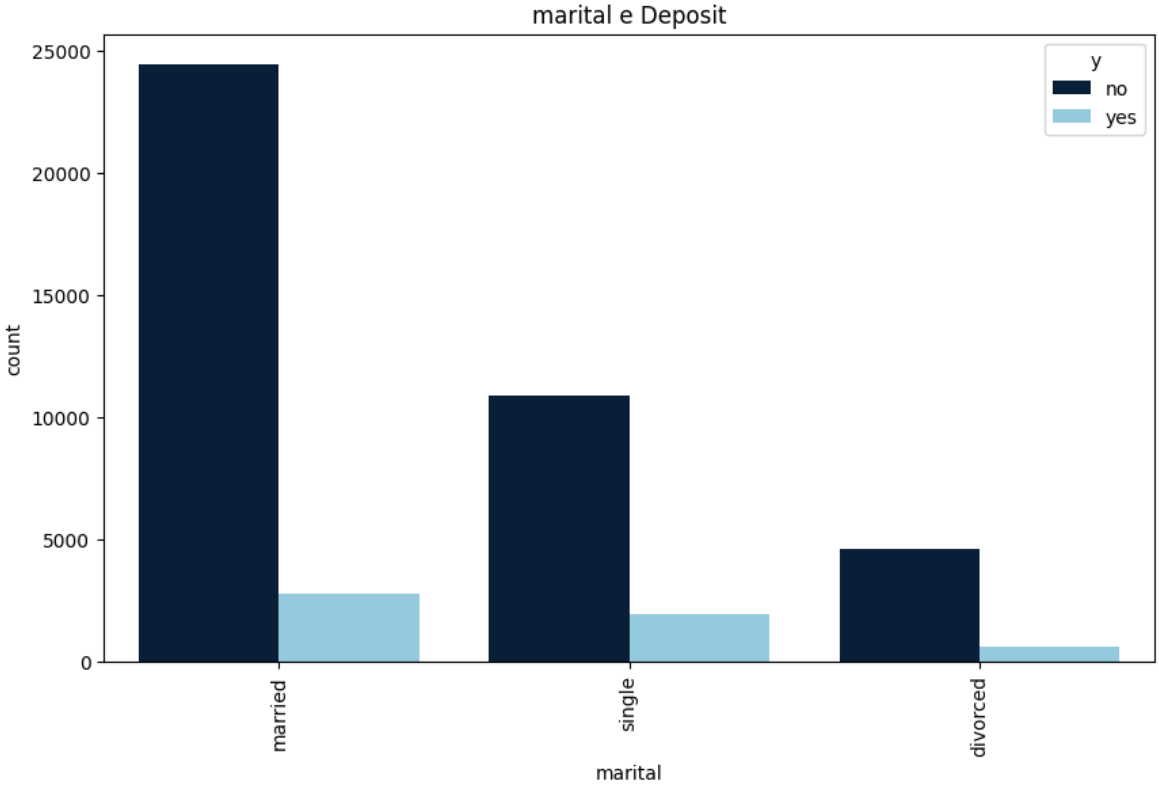
ANALISI BIVARIATA DELLA VARIABILE DI RISPOSTA

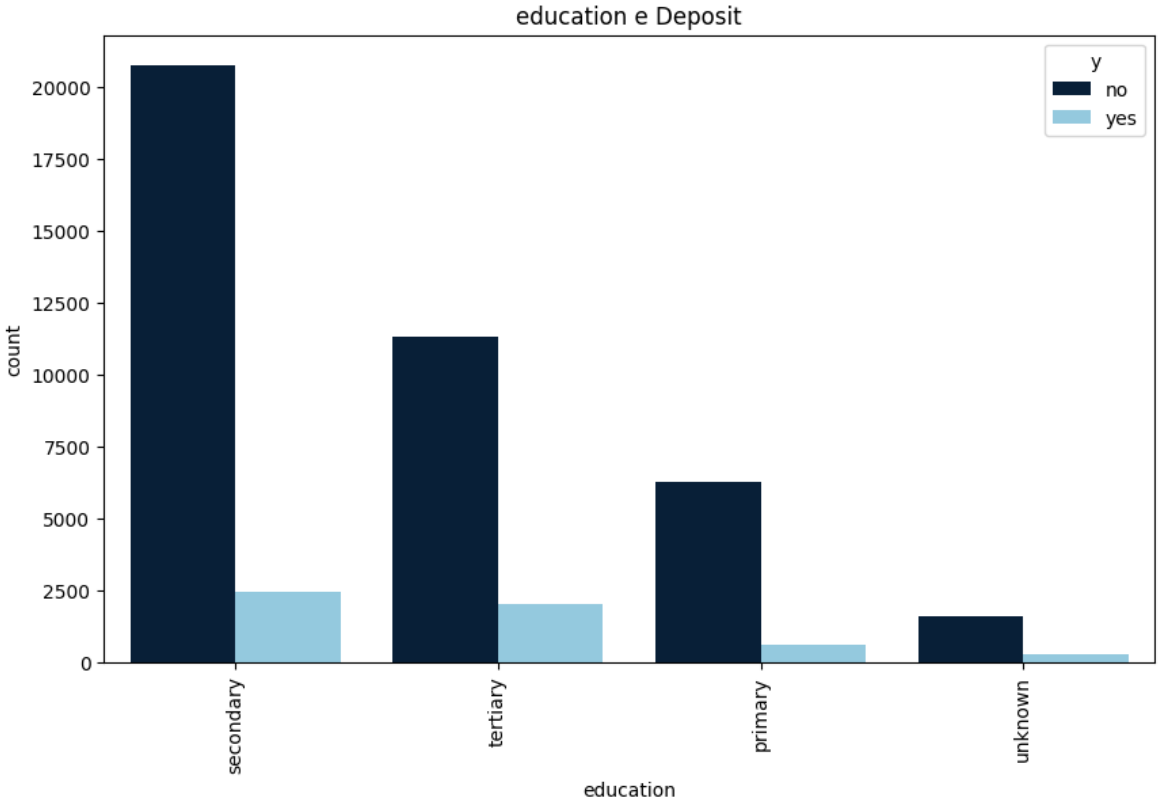
Variabili Categoriche

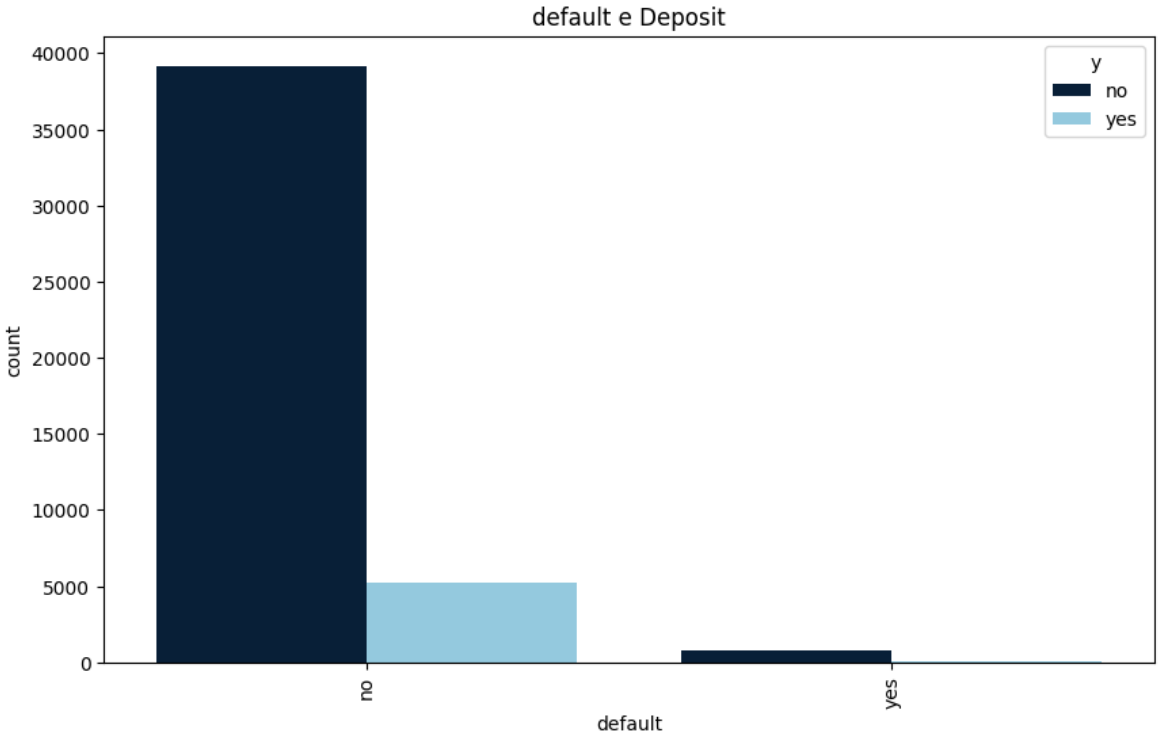
In [ ]:
```
value_counts = df_wrgld['y'].value_counts()
value_counts.plot.bar(title = 'Sottoscrizioni Conto Deposito')

colors = sns.color_palette(['#001F3F', '#87CEEB'])
for var in categorical_variables:
    plt.figure(figsize=(10, 6))
    sns.countplot(x=var, hue='y', data=df_wrgld, order=df_wrgld[var].value_count
    plt.title(f'{var} e Deposit')
    plt.xticks(rotation=90)
    plt.xlabel(var)
    plt.show()
```

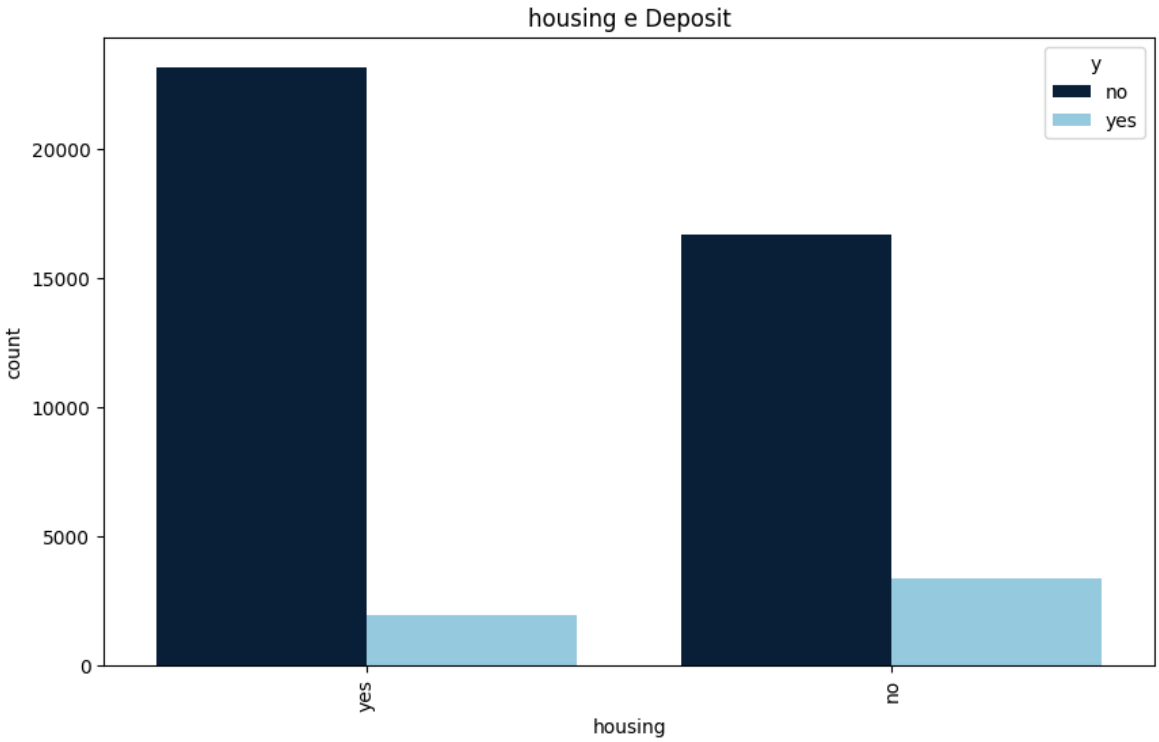

Sottoscrizioni Conto Deposito
40000
35000
30000
25000
20000
15000
10000
5000
0
no
yes
y

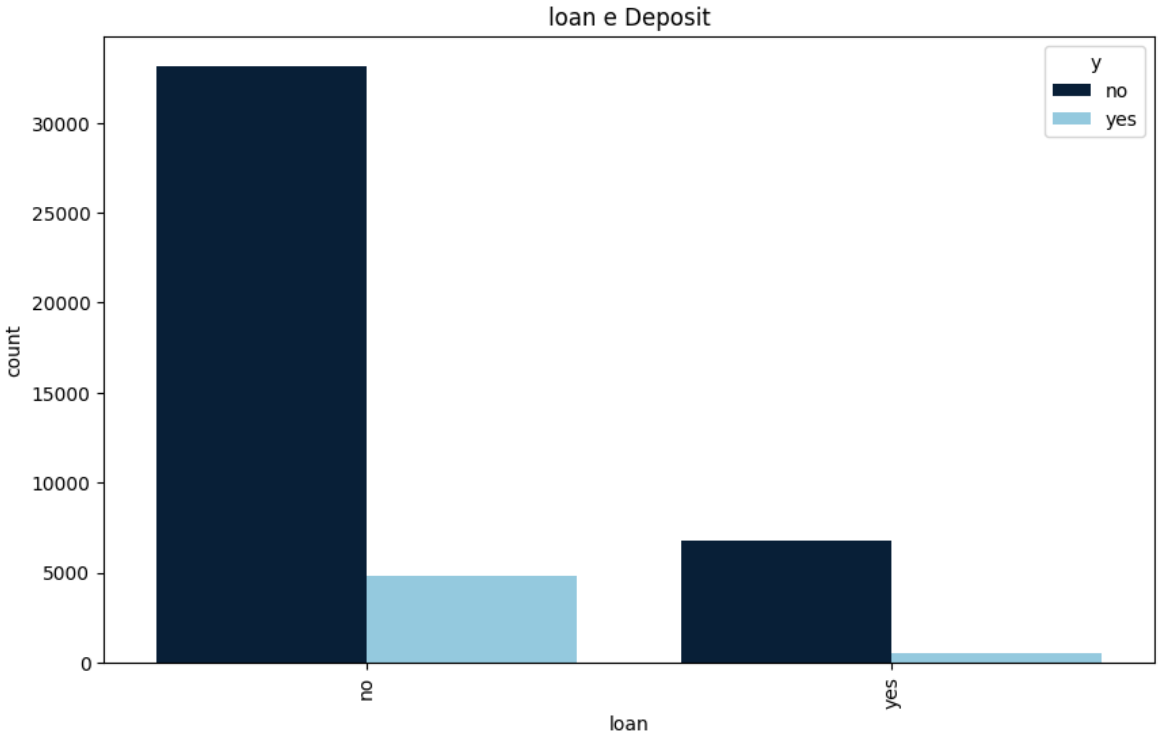

job e Deposit
y
no
yes
count
8000
6000
4000
2000
0
blue-collar
management
technician
admin.
services
retired
self-employed
entrepreneur
unemployed
housemaid
student
unknown
job

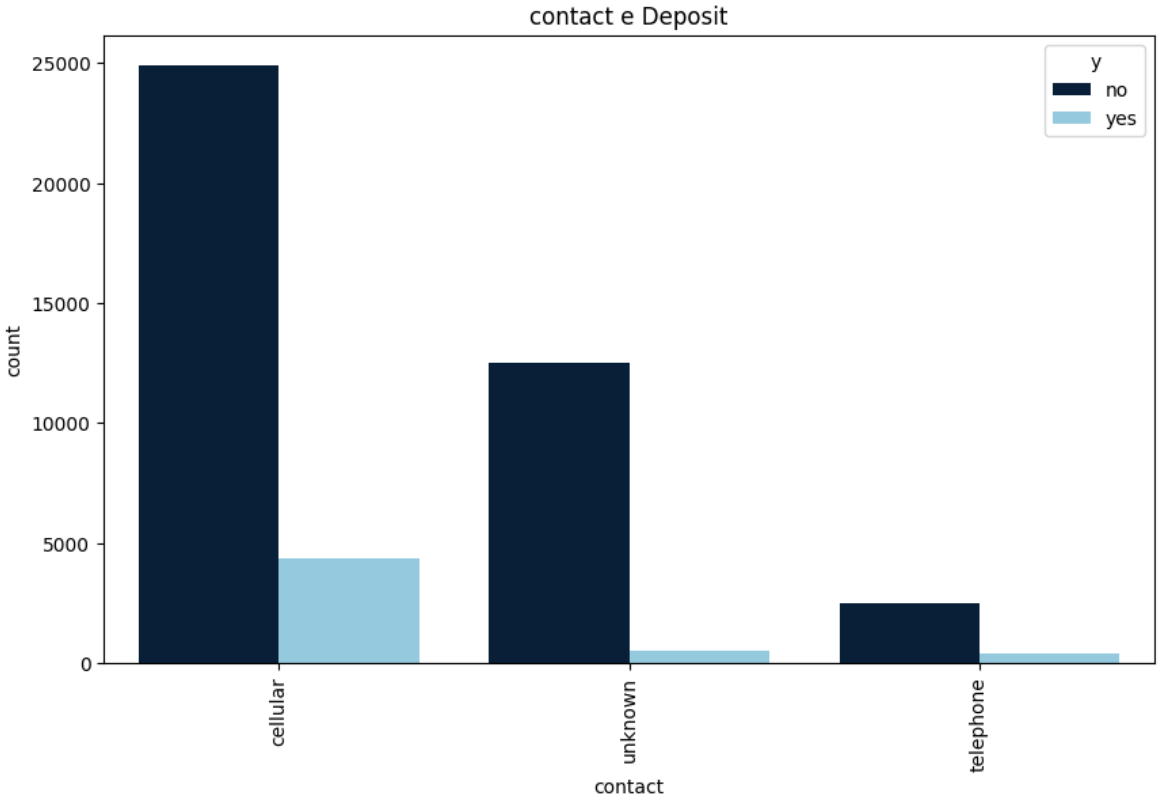

marital e Deposit
y
no
yes
count
25000
20000
15000
10000
5000
0
married
single
divorced
marital

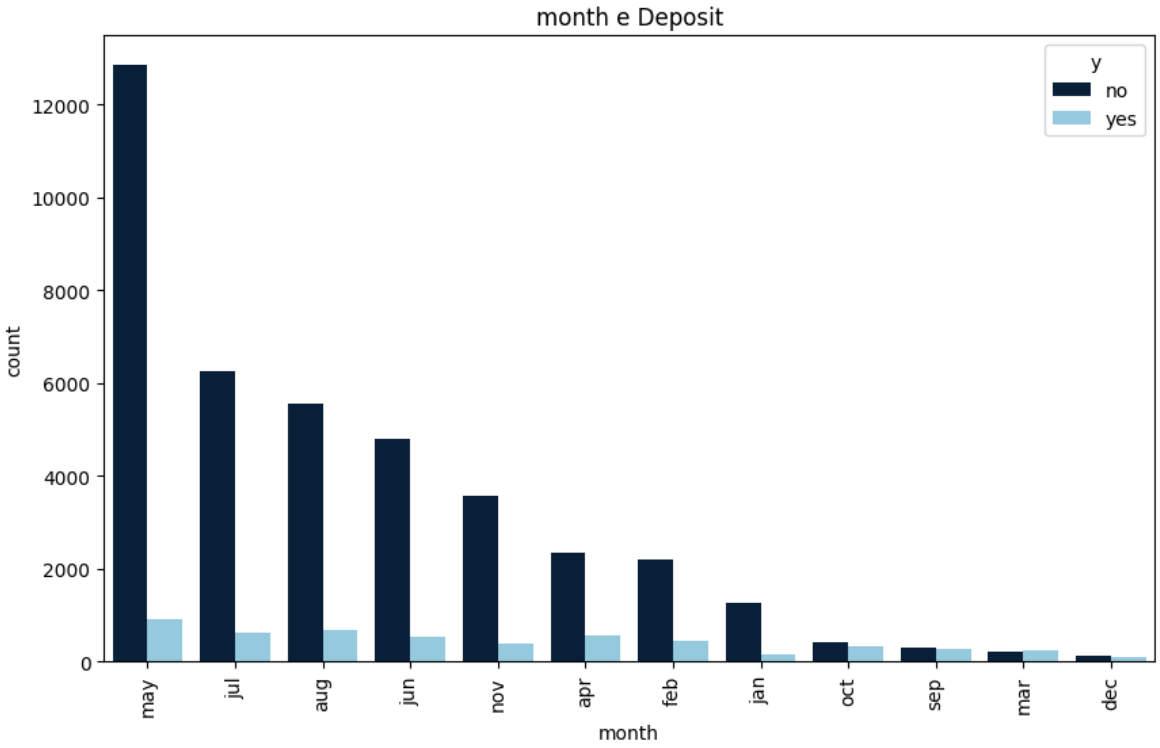

education e Deposit
y
no
yes
count
20000
17500
15000
12500
10000
7500
5000
2500
0
secondary
tertiary
primary
unknown
education

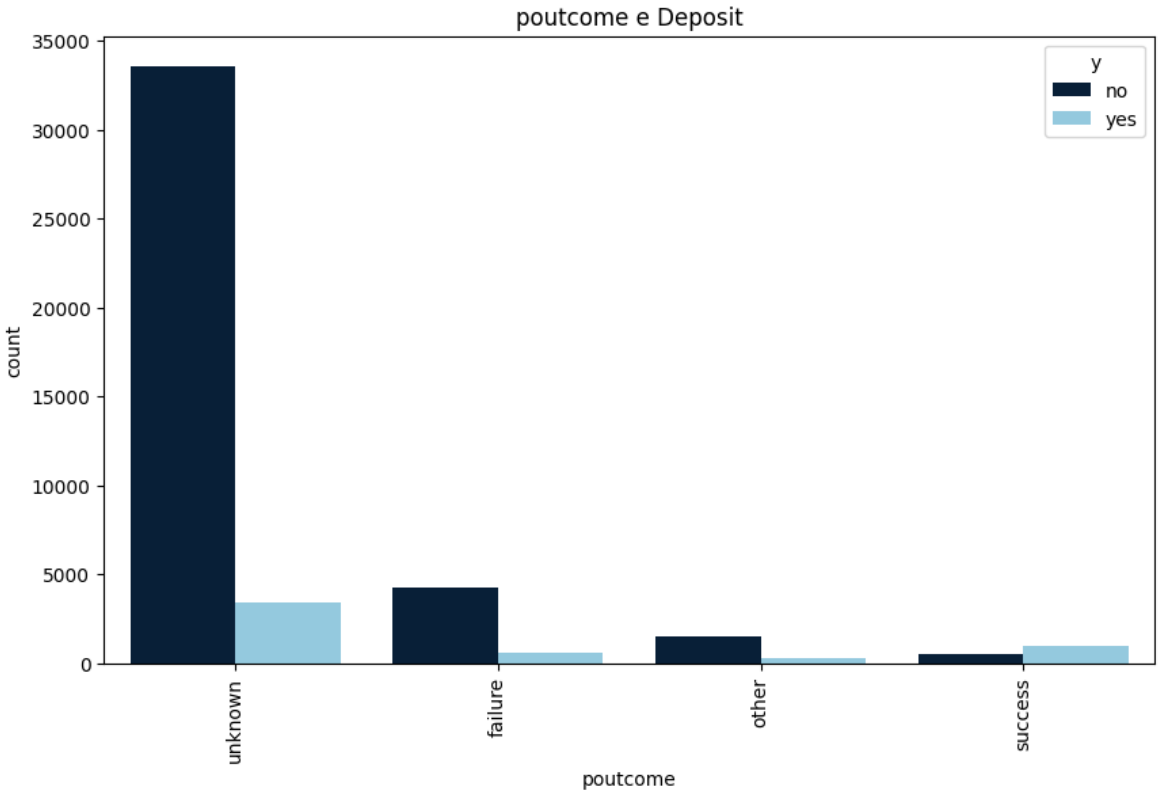

default e Deposit
y
no
yes
40000
35000
30000
25000
20000
15000
10000
5000
0
count
no
yes
default


housing e Deposit
y
no
yes
20000
15000
10000
5000
0
count
yes
no
housing


loan e Deposit
y
no
yes
30000
25000
20000
15000
10000
5000
0
count
no
yes
loan


contact e Deposit
y
no
yes
25000
20000
15000
10000
5000
0
count
cellular
unknown
telephone
contact



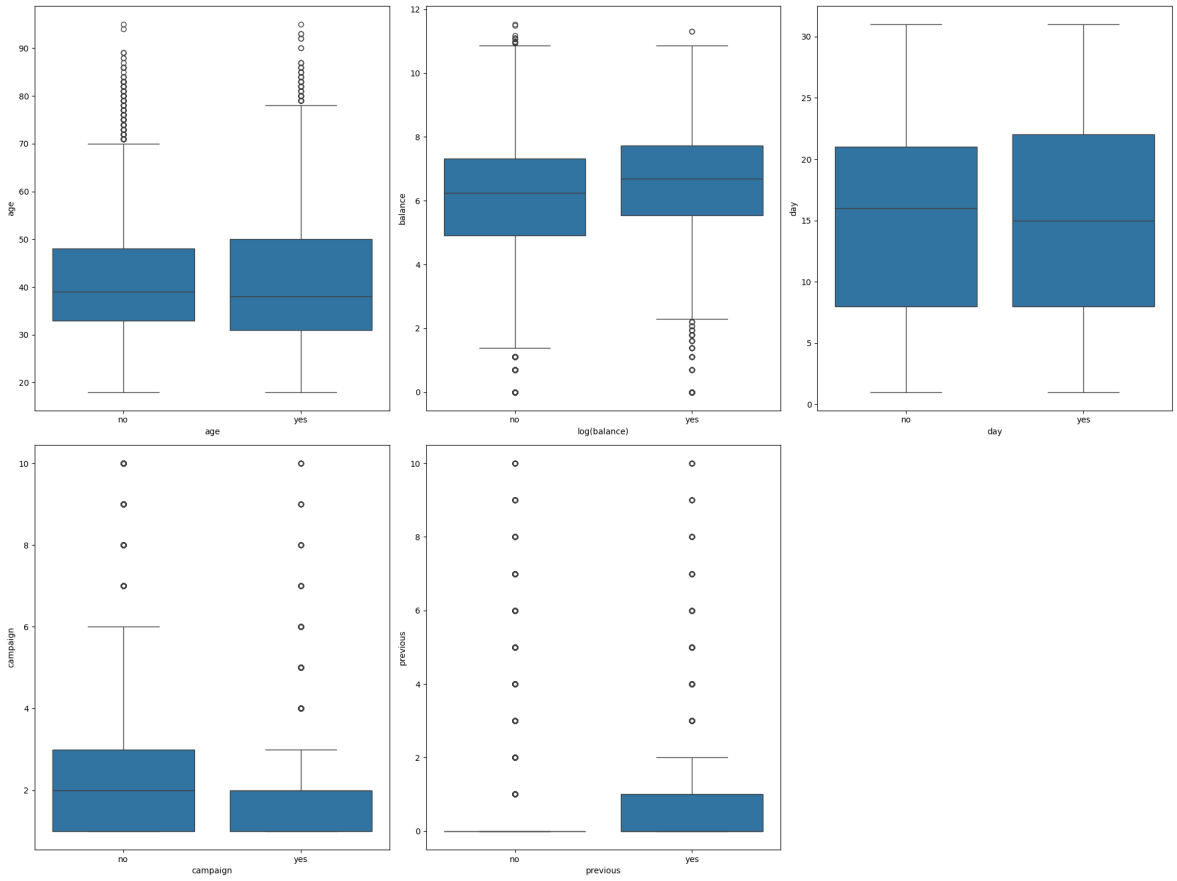


Variabili Numeriche

In [ ]:
```
numerical_variables = ['age', 'balance', 'day', 'campaign', 'previous']

plt.figure(figsize=(20, 15), facecolor='white')

for i, feature in enumerate(numerical_variables, 1):
    plt.subplot(2, 3, i)

    if feature == 'balance':
        sns.boxplot(x="y", y=np.log1p(df_wrgld[feature]), data=df_wrgld)
        plt.xlabel(f'log({feature})')
```

```
    else:
        sns.boxplot(x="y", y=df_wrgld[feature], data=df_wrgld)
        plt.xlabel(feature)

plt.tight_layout()
plt.show()
```

```
C:\Users\cottu\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2
kfra8p0\LocalCache\local-packages\Python311\site-packages\pandas\core\arraylike.p
y:396: RuntimeWarning: divide by zero encountered in log1p
  result = getattr(ufunc, method)(*inputs, **kwargs)
C:\Users\cottu\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2
kfra8p0\LocalCache\local-packages\Python311\site-packages\pandas\core\arraylike.p
y:396: RuntimeWarning: invalid value encountered in log1p
  result = getattr(ufunc, method)(*inputs, **kwargs)
```



PREPARAZIONE DEL DATASET E "ENCODING" Prima di applicare le tecniche di Machine Learning, trasformo le colonne categoriche in variabili numeriche (One-Hot-Encoding) e le variabili categoriche binarie (“yes” e “no”) in colonne che contengano solo valori booleani.

Importo le librerie necessarie all'implementazione di modelli di Machine Learning

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sc
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
```

```
from sklearn.pipeline import Pipeline
from xgboost import XGBClassifier
```

Separo le features dalla variabile target ("y") e definisco le colonne numeriche e categoriche

In [ ]:
```
X = df_wrgld.drop('y', axis=1)
y = df_wrgld['y']
print(df_wrgld.columns.tolist())
numeric_features = ['age', 'balance', 'day', 'campaign', 'previous']
categorical_features = ['job', 'marital', 'education', 'default', 'housing', 'lo
```

```
['age', 'job', 'marital', 'education', 'default', 'balance', 'housing', 'loan',
'contact', 'day', 'month', 'campaign', 'previous', 'poutcome', 'y']
```

Definisco i rispettivi trasformatori per le variabili numeriche e categoriche

In [ ]:
```
numeric_transformer = Pipeline(steps=[
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ('onehot', OneHotEncoder())
])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])
```

Creo i seguenti tre modelli di ML:

- Logistic Regression
- Decision Tree
- Random Forest

In [ ]:
```
logistic_model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression(random_state=42))
])

decision_tree_model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', DecisionTreeClassifier(random_state=42))
])

random_forest_model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', RandomForestClassifier(random_state=100, n_jobs = -1))
])


models = {
    'Logistic Regression': logistic_model,
    'Decision Tree': decision_tree_model,
```

```
    'Random Forest': random_forest_model
}
```

Divido i dati in set di addestramento e di test e addestro i tre modelli

In [ ]:
```
for model_name, model in models.items():
    print(f"\n{model_name}:\n")

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran

    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, pos_label='yes')
    recall = recall_score(y_test, y_pred, pos_label='yes')
    f1 = f1_score(y_test, y_pred, pos_label='yes')

    print(f'Accuracy: {accuracy:.4f}')
    print(f'Precision: {precision:.4f}')
    print(f'Recall: {recall:.4f}')
    print(f'F1 Score: {f1:.4f}')
```

```
Logistic Regression:

C:\Users\cottu\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2
kfra8p0\LocalCache\local-packages\Python311\site-packages\sklearn\linear_model\_l
ogistic.py:460: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Accuracy: 0.8902
Precision: 0.6644
Recall: 0.1815
F1 Score: 0.2851

Decision Tree:

Accuracy: 0.8256
Precision: 0.2923
Recall: 0.3135
F1 Score: 0.3025

Random Forest:

Accuracy: 0.8903
Precision: 0.6170
Recall: 0.2392
F1 Score: 0.3448
```

Il modello Random Forest presenta un F1-score migliore degli altri modelli. Nel nostro caso specifico, la variabile risposta mostra uno sbilanciamento nella proporzione dei valori per i due risultati della campagna ("yes" e "no"), con una predominanza di uno dei

due. Quando si affronta uno sbilanciamento così marcato, la metrica di valutazione dell'F1-score può essere più informativa rispetto all'accuratezza (accuracy).

L'F1-score è una metrica che tiene conto sia della precisione che del richiamo. Questi due concetti sono particolarmente rilevanti quando si gestiscono classi sbilanciate. Per chiarire: Precisione (Precision): Indica la proporzione di istanze predette come positive che sono effettivamente positive. In termini di una campagna di marketing, sarebbe la percentuale di casi predetti come "yes" che sono effettivamente "yes".

Richiamo (Recall): Indica la proporzione di istanze positive effettive che sono state correttamente predette. Nel contesto della campagna, sarebbe la percentuale di casi "yes" che sono stati identificati correttamente.

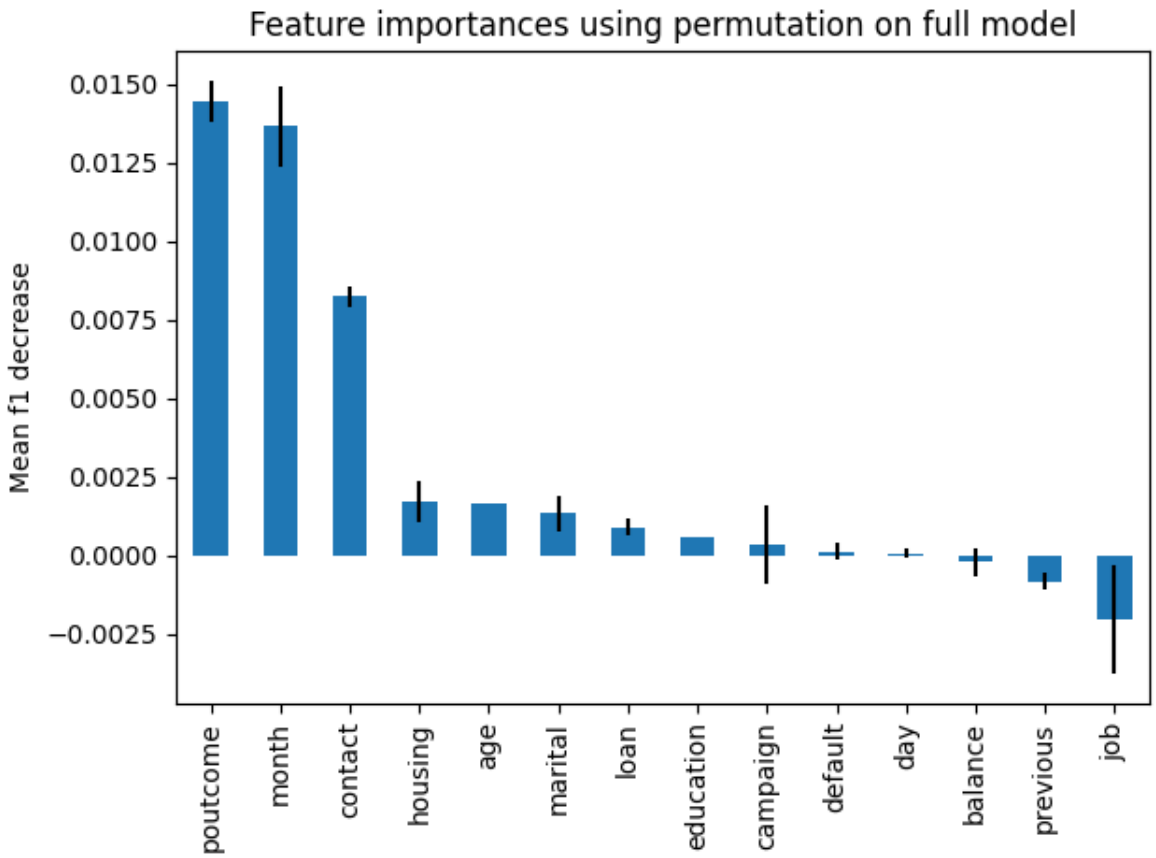
Quando si ha uno sbilanciamento e la classe di interesse è in minoranza (come spesso accade in casi di previsione di eventi come conversioni in una campagna di marketing), la precisione da sola può essere ingannevole. Ad esempio, se il modello predice tutte le istanze come "no" in un caso di forte sbilanciamento, potrebbe ancora ottenere un'alta accuracy, ma fallirebbe completamente nel rilevare la classe di interesse.

L'F1-score bilancia precisione e richiamo, fornendo una singola metrica che tiene conto di entrambi. In caso di sbilanciamento, un modello che predice sempre la classe maggioritaria otterrebbe un punteggio F1 basso. Pertanto, in presenza di sbilanciamento, l'F1-score è spesso preferito rispetto all'accuratezza per fornire una valutazione più completa delle prestazioni del modello.

Una volta identificato il modello Random Forest come quello che meglio si adatta ai dati ed offre una maggiore capacità predittiva, identifico quali sono le features che hanno la maggiore importanza per la previsione dei risultati della campagna. Ho scelto di implementare la Permutation Feature Importance, il quale metodo valuta come la precisione del modello varia quando si mescolano casualmente i valori di una singola caratteristica, misurando così l'impatto di quella caratteristica sulla capacità predittiva del modello. L'idea fondamentale è che se la permutazione di una particolare caratteristica causa una significativa diminuzione delle prestazioni del modello, allora quella caratteristica è importante per il modello.

In [ ]:
```
from sklearn.inspection import permutation_importance

rf_model = random_forest_model.fit(X_train, y_train)

result = permutation_importance(
    rf_model, X_test, y_test, n_repeats=2, random_state=42, n_jobs=-1, scoring =
)

feature_names = X.columns
forest_importances = pd.Series(result.importances_mean, index=feature_names)

forest_importances_sorted = forest_importances.sort_values(ascending=False)

fig, ax = plt.subplots()
forest_importances_sorted.plot.bar(yerr=result.importances_std, ax=ax)
ax.set_title("Permutation Feature Importance")
```

```
ax.set_ylabel("Mean f1 decrease")
fig.tight_layout()
plt.show()
```



Le variabili che maggiormente influenzano la performance del modello e, di conseguenza, il successo della campagna di marketing sono:

- poutcome, che rappresenta il risultato della campagna di marketing precedente, indicando che coloro che hanno sottoscritto un conto deposito precedentemente, tendenzialmente sono inclini a riacquistarlo;
- month, indicando l'esistenza di una certa stagionalità per il risultato della campagna di marketing;
- contact, il tipo di contatto influisce particolarmente sulla bontà del modello, nel nostro caso il cellulare è stato il mezzo che ha ottenuro una maggiore conversione positiva della campagna.