Ensemble Project - Term Deposit Subscription

March 16, 2021

1 Attribute Information

1 - age

2 - job : type of job

3 - marital : marital status

4 - education

5 - default: has credit in default?

6 - housing: has housing loan?

7 - loan: has personal loan?

8 - balance in account

9 - contact: contact communication type

10 - month: last contact month of year

11 - day: last contact day of the month

12 - duration: last contact duration, in seconds

13 - campaign: number of contacts performed during this campaign and for this client

14 - pdays: number of days that passed by after the client was last contacted from a previous campaign

15 - previous: number of contacts performed before this campaign and for this client

16 - poutcome: outcome of the previous marketing campaign

17 - Output variable ('Target'): has the client subscribed a term deposit?

2 Problem statement (Term Deposit Sale)

We have data from a Portuguese bank on details of customers related to selling a term deposit. The objective of the project is to help the marketing team identify potential customers who are relatively more likely to subscribe to the term deposit and this increase the hit ratio.

What is a Term Deposit? A Term deposit is a deposit that a bank or a financial institution offers with a fixed rate (often better than just opening deposit account) in which your money will be returned back at a specific maturity time.

[1]:
```python
import pandas as pd
import numpy as np
import seaborn as sns
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import recall_score, roc_auc_score,␣
 ↪classification_report,confusion_matrix,accuracy_score,precision_score,f1_score
from sklearn.ensemble import RandomForestClassifier,␣
 ↪AdaBoostClassifier,GradientBoostingClassifier,BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

sns.set()
df_in = pd.read_csv('bank-full.csv')
df_in.head()
```

[1]:
```
   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 Target
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
```

3 Deliverable –1 (Exploratory data analysis)–(15)

3.0.1 Univariate analysis (9marks)

Data types and description of the independent attributes which should include (name, range of values observed, central values (mean and median), standard deviation and quartiles, skewness). - 3 Marks

[2]:
```python
#remove unknown values from education
df_in.loc[df_in['education'] == 'unknown', ['education']] = 'secondary'
#remove unknown values from contact
df_in.loc[df_in['contact'] == 'unknown', ['contact']] = 'telephone'
#remove unknown values from job
df_in.loc[df_in['job'] == 'unknown', ['job']] = 'blue-collar'
```

[3]:
```python
for col in df_in.columns:
    print(col + ':')
    print(df_in[col].unique())
```

```
    print(df_in[col].nunique())
```

```
age:
[58 44 33 47 35 28 42 43 41 29 53 57 51 45 60 56 32 25 40 39 52 46 36 49
 59 37 50 54 55 48 24 38 31 30 27 34 23 26 61 22 21 20 66 62 83 75 67 70
 65 68 64 69 72 71 19 76 85 63 90 82 73 74 78 80 94 79 77 86 95 81 18 89
 84 87 92 93 88]
77
job:
['management' 'technician' 'entrepreneur' 'blue-collar' 'retired' 'admin.'
 'services' 'self-employed' 'unemployed' 'housemaid' 'student']
11
marital:
['married' 'single' 'divorced']
3
education:
['tertiary' 'secondary' 'primary']
3
default:
['no' 'yes']
2
balance:
[ 2143    29     2 …  8205 14204 16353]
7168
housing:
['yes' 'no']
2
loan:
['no' 'yes']
2
contact:
['telephone' 'cellular']
2
day:
[ 5  6  7  8  9 12 13 14 15 16 19 20 21 23 26 27 28 29 30  2  3  4 11 17
 18 24 25  1 10 22 31]
31
month:
['may' 'jun' 'jul' 'aug' 'oct' 'nov' 'dec' 'jan' 'feb' 'mar' 'apr' 'sep']
12
duration:
[ 261  151   76 … 1298 1246 1556]
1573
campaign:
[ 1  2  3  5  4  6  7  8  9 10 11 12 13 19 14 24 16 32 18 22 15 17 25 21
 43 51 63 41 26 28 55 50 38 23 20 29 31 37 30 46 27 58 33 35 34 36 39 44]
48
pdays:
```

```
[ -1 151 166  91  86 143 147  89 140 176 101 174 170 167 195 165 129 188
 196 172 118 119 104 171 117 164 132 131 123 159 186 111 115 116 173 178
 110 152  96 103 150 175 193 181 185 154 145 138 126 180 109 158 168  97
 182 127 130 194 125 105 102  26 179  28 183 155 112 120 137 124 187 190
 113 162 134 169 189   8 144 191 184 177   5  99 133  93  92  10 100 156
 198 106 153 146 128   7 121 160 107  90  27 197 136 139 122 157 149 135
  30 114  98 192 163  34  95 141  31 199  94 108  29 268 247 253 226 244
 239 245 204 231 238 258 230 254 265  71 223 246 250 266 240 205 261 259
 241 260 234 251 225 161 237 262 248 255 220 227 206 224 249 235 228 263
   2 270 232 252 207 200 269 233 256 273 272 242 264 208 214 222 271 203
 221 202 216 201 257 229 210 217  75 213  73  76 267 211 215  77 236  82
   6 209 274   1 243 212 275  80 276   9 279  12 280  88 277  85  84 219
  24  21 282  41 294  49 329 307 303 331 308 300  64 314 287 330 332 302
 323 318 333  60 326 335 313 312 305 325 327 336 309 328 322  39 316 292
 295 310 306 320 317 289  57 321 142 339 301 315 337 334 340 319  17  74
 148 341 299 344 342 324 345 346 304 281 343 338  14 347  15 291 348 349
 285 350 284  25 283 278  81   4  87  83  79  70  13 293  37  78  63  22
 296 355  66  19  35 360 357 354 351 362 358 365 298 286 364 363  47 361
 288 366 356 352 359 297 367 353 368  42 290  67 371 370 369  50  36 373
 374 372 311 375 378  59 379  40  18  43  20  69  38 385  56  55  44 391
  72 390  32  62 399 393  65 377 395 388 389 386  61 412 405 434 394 382
 459 440 397 383  68 461 462 463 422  51 457 430 442 403 454 428 392 410
 401 474 475 477 478  54 476 380 479  45  46 495  58  48 518  52 515 520
 511 536 387 218  33 544 435 436 555 433 446 558 469 616 561 553 384 592
 467 585 480 421 667 626 426 595 381 376 648 521 452 449 633 398  53 460
 670 551 414 557 687 404 651 686 425 504 578 674 416 586 411 756 450 745
 514 417 424 776 396 683 529 439 415 456 407 458 532 481 791 701 531 792
 413 445 535 784 419 455 491 431 542 470 472 717 437   3 782 728 828 524
 562 761 492 775 579 493 464 760 466 465 656 831 490 432 655 427 749 838
 769 587 778 854 779 850 771 594 842 589 603 484 489 486 409 444 680 808
 485 503 690 772 774 526 420 528 500 826 804 508 547 805 541 543 871 550
 530]
559
previous:
[  0   3   1   4   2  11  16   6   5  10  12   7  18   9  21   8  14  15
  26  37  13  25  20  27  17  23  38  29  24  51 275  22  19  30  58  28
  32  40  55  35  41]
41
poutcome:
['unknown' 'failure' 'other' 'success']
4
Target:
['no' 'yes']
2
```

[4]:
```
#convert yes/no to 1/0
cols = ['Target', 'loan', 'housing', 'default', 'marital']
```

```
df = pd.get_dummies(df_in, columns=cols, drop_first=True)
df.head()
```

[4]:
```
   age           job  education  balance    contact  day month  duration  \
0   58    management   tertiary     2143  telephone    5   may       261
1   44    technician  secondary       29  telephone    5   may       151
2   33  entrepreneur  secondary        2  telephone    5   may        76
3   47   blue-collar  secondary     1506  telephone    5   may        92
4   33   blue-collar  secondary        1  telephone    5   may       198

   campaign  pdays  previous poutcome  Target_yes  loan_yes  housing_yes  \
0         1     -1         0  unknown           0         0            1
1         1     -1         0  unknown           0         0            1
2         1     -1         0  unknown           0         1            1
3         1     -1         0  unknown           0         0            1
4         1     -1         0  unknown           0         0            0

   default_yes  marital_married  marital_single
0            0                1               0
1            0                0               1
2            0                1               0
3            0                1               0
4            0                0               1
```

[5]:
```
#secondary = 'secondary'
#contact unknown:

df.head()
```

[5]:
```
   age           job  education  balance    contact  day month  duration  \
0   58    management   tertiary     2143  telephone    5   may       261
1   44    technician  secondary       29  telephone    5   may       151
2   33  entrepreneur  secondary        2  telephone    5   may        76
3   47   blue-collar  secondary     1506  telephone    5   may        92
4   33   blue-collar  secondary        1  telephone    5   may       198

   campaign  pdays  previous poutcome  Target_yes  loan_yes  housing_yes  \
0         1     -1         0  unknown           0         0            1
1         1     -1         0  unknown           0         0            1
2         1     -1         0  unknown           0         1            1
3         1     -1         0  unknown           0         0            1
4         1     -1         0  unknown           0         0            0

   default_yes  marital_married  marital_single
0            0                1               0
1            0                0               1
2            0                1               0
```

```
3               0                    1                  0
4               0                    0                  1
```

[6]:
```
#df.drop('pdays', axis=1, inplace=True)
```

[7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 18 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   age              45211 non-null  int64
 1   job              45211 non-null  object
 2   education        45211 non-null  object
 3   balance          45211 non-null  int64
 4   contact          45211 non-null  object
 5   day              45211 non-null  int64
 6   month            45211 non-null  object
 7   duration         45211 non-null  int64
 8   campaign         45211 non-null  int64
 9   pdays            45211 non-null  int64
 10  previous         45211 non-null  int64
 11  poutcome         45211 non-null  object
 12  Target_yes       45211 non-null  uint8
 13  loan_yes         45211 non-null  uint8
 14  housing_yes      45211 non-null  uint8
 15  default_yes      45211 non-null  uint8
 16  marital_married  45211 non-null  uint8
 17  marital_single   45211 non-null  uint8
dtypes: int64(7), object(5), uint8(6)
memory usage: 4.4+ MB
```

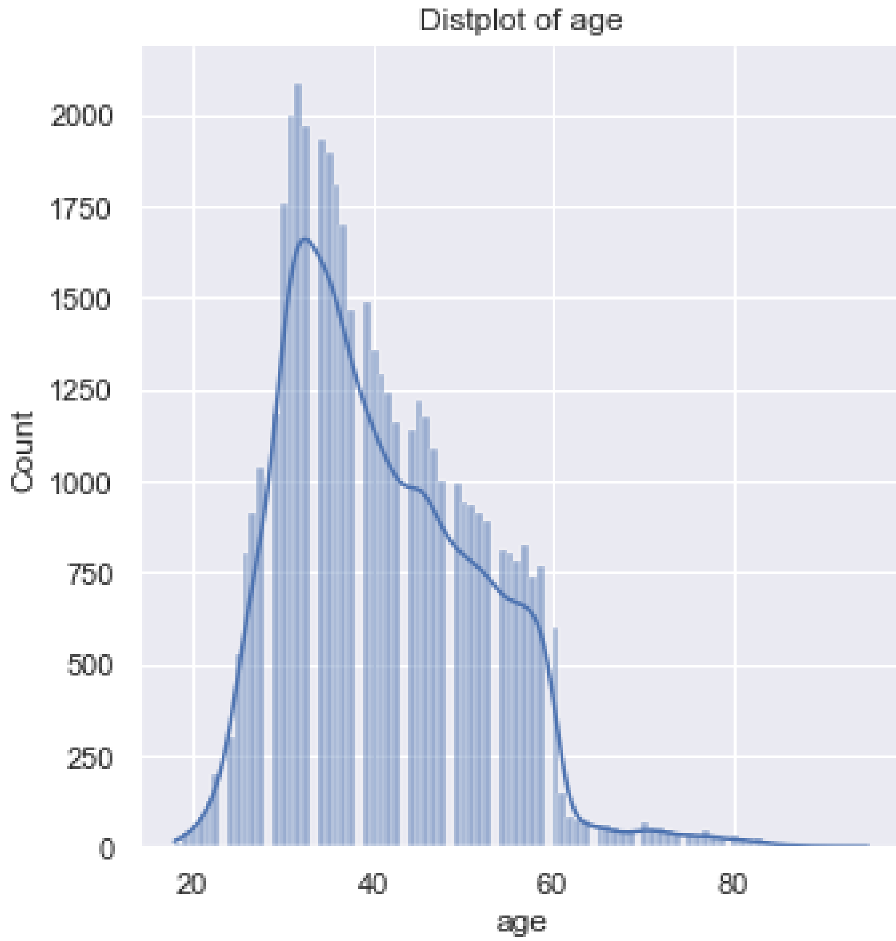
[8]:
```
df.describe().T
```

[8]:
```
                   count         mean          std     min    25%    50%  \
age              45211.0    40.936210    10.618762    18.0   33.0   39.0
balance          45211.0  1362.272058  3044.765829 -8019.0   72.0  448.0
day              45211.0    15.806419     8.322476     1.0    8.0   16.0
duration         45211.0   258.163080   257.527812     0.0  103.0  180.0
campaign         45211.0     2.763841     3.098021     1.0    1.0    2.0
pdays            45211.0    40.197828   100.128746    -1.0   -1.0   -1.0
previous         45211.0     0.580323     2.303441     0.0    0.0    0.0
Target_yes       45211.0     0.116985     0.321406     0.0    0.0    0.0
loan_yes         45211.0     0.160226     0.366820     0.0    0.0    0.0
housing_yes      45211.0     0.555838     0.496878     0.0    0.0    1.0
default_yes      45211.0     0.018027     0.133049     0.0    0.0    0.0
marital_married  45211.0     0.601933     0.489505     0.0    0.0    1.0
```

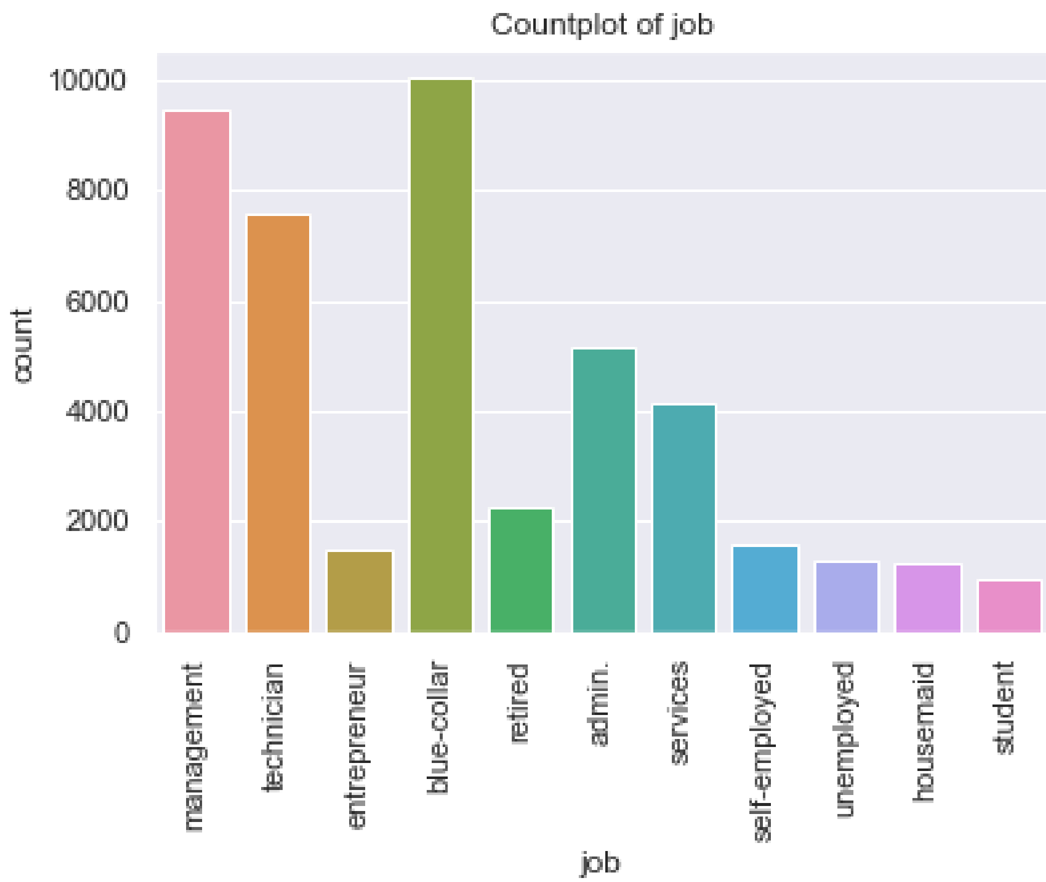
```
marital_single   45211.0     0.282896     0.450411     0.0     0.0     0.0

                    75%       max
age                48.0      95.0
balance          1428.0  102127.0
day                21.0      31.0
duration          319.0    4918.0
campaign            3.0      63.0
pdays              -1.0     871.0
previous            0.0     275.0
Target_yes          0.0       1.0
loan_yes            0.0       1.0
housing_yes         1.0       1.0
default_yes         0.0       1.0
marital_married     1.0       1.0
marital_single      1.0       1.0
```

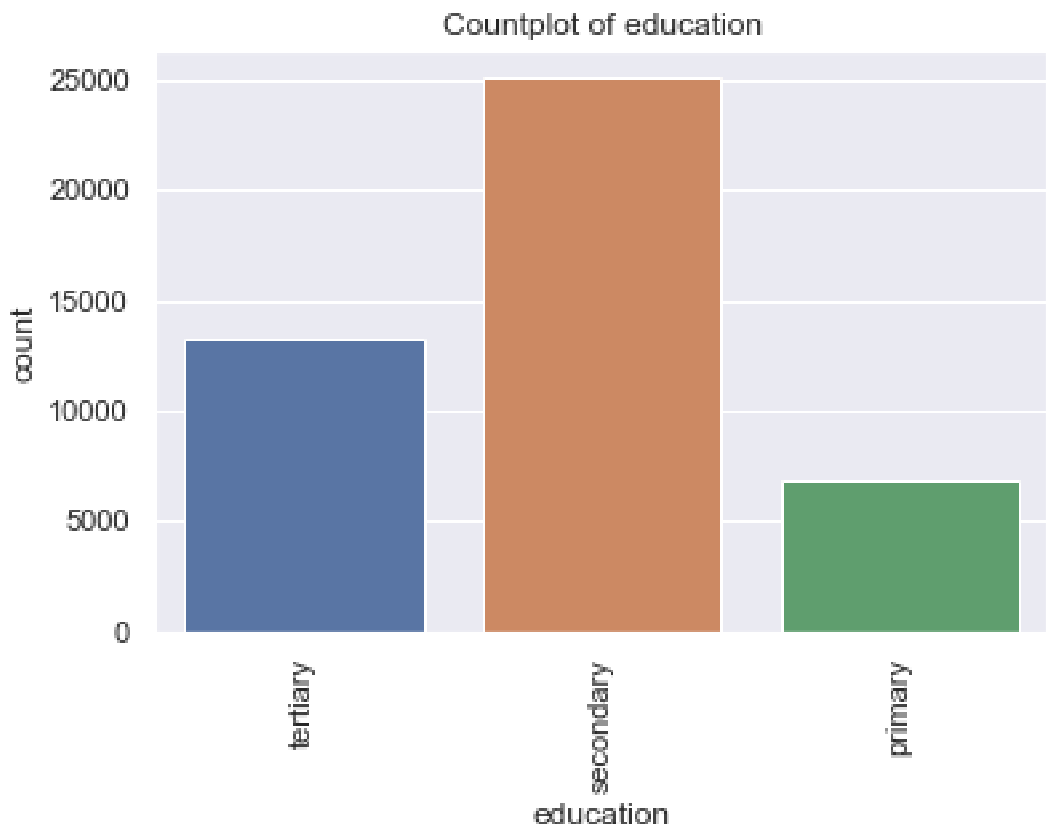
Make a function to plot 'countplot' if the variable is categorical and 'distplot' if the variable is numeric. - 3 Marks Identify outliers using IQR and verify the same using plots. Also mention the percentage of data points which are considered outliers. Should we treat them, why or why not? - 3 Marks

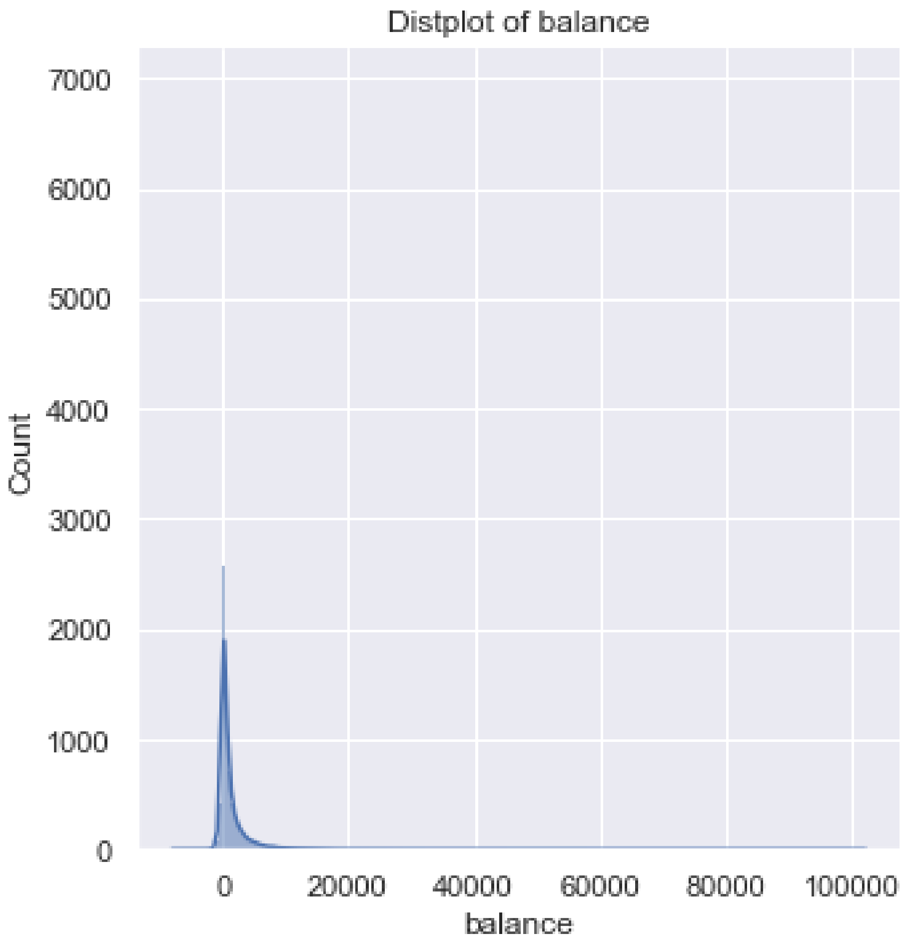
[9]:
```python
for col in df.columns:
    if df[col].dtypes == 'object':
        sns.countplot(data=df, x=col)
        temp = 'Countplot of '+col
        plt.title(temp)
        plt.xticks(rotation=90)
        plt.show()
    else:
        sns.displot(data=df, x=col, kde=True)
        temp = 'Distplot of '+col
        plt.title(temp)
        plt.show()
```

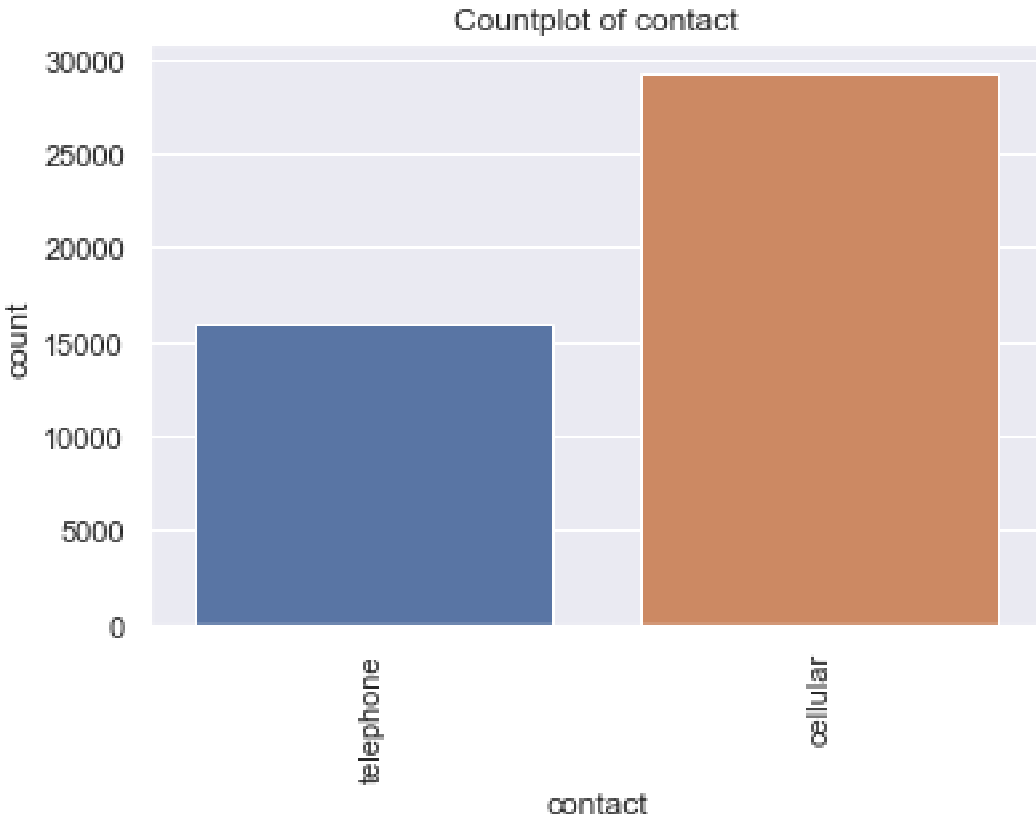


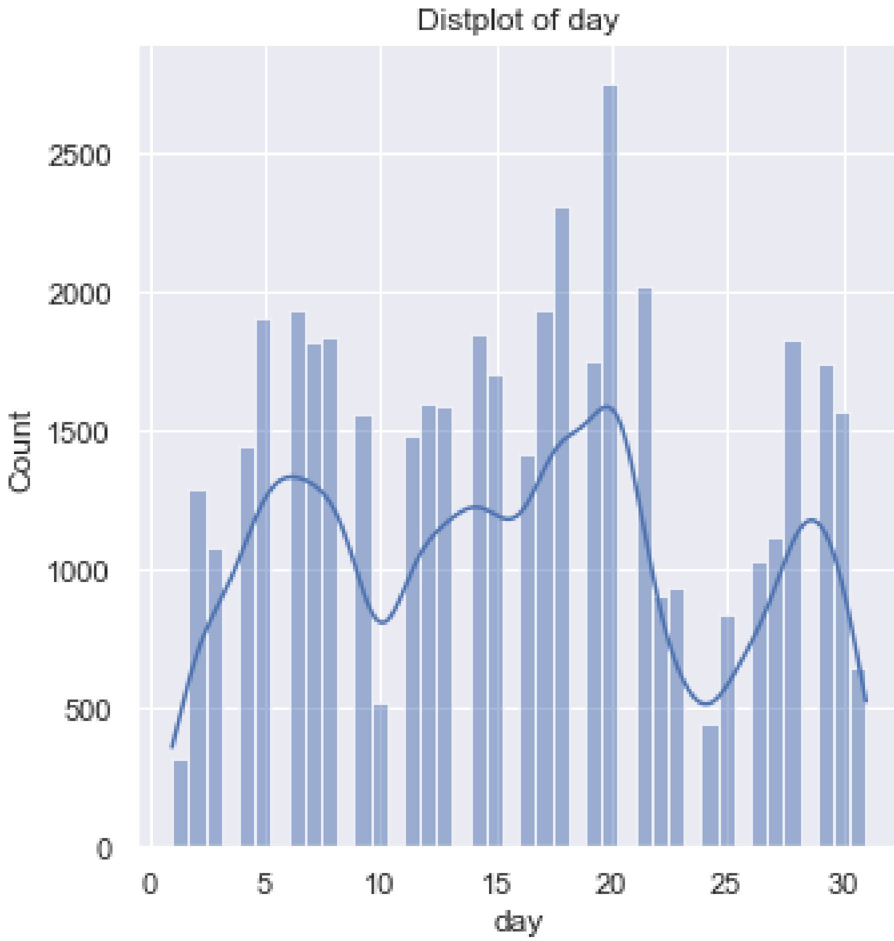
Distplot of age
Count
2000
1750
1500
1250
1000
750
500
250
0
20
40
60
80
age

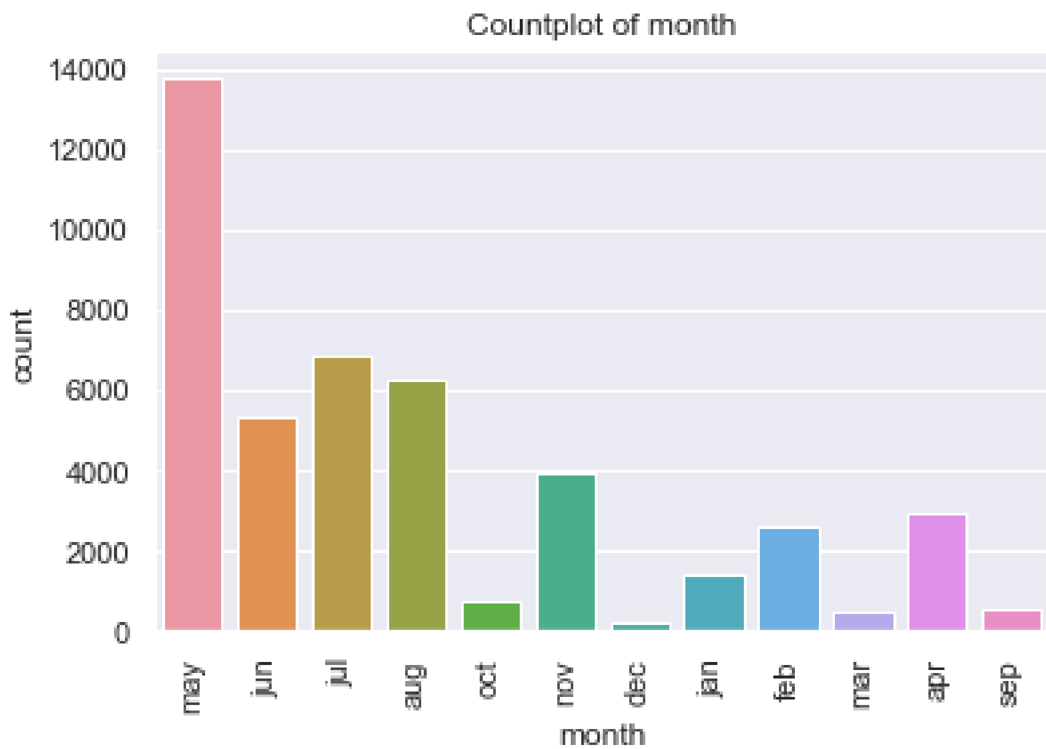

Countplot of job
count
10000
8000
6000
4000
2000
0
management
technician
entrepreneur
blue-collar
retired
admin.
services
self-employed
unemployed
housemaid
student
job

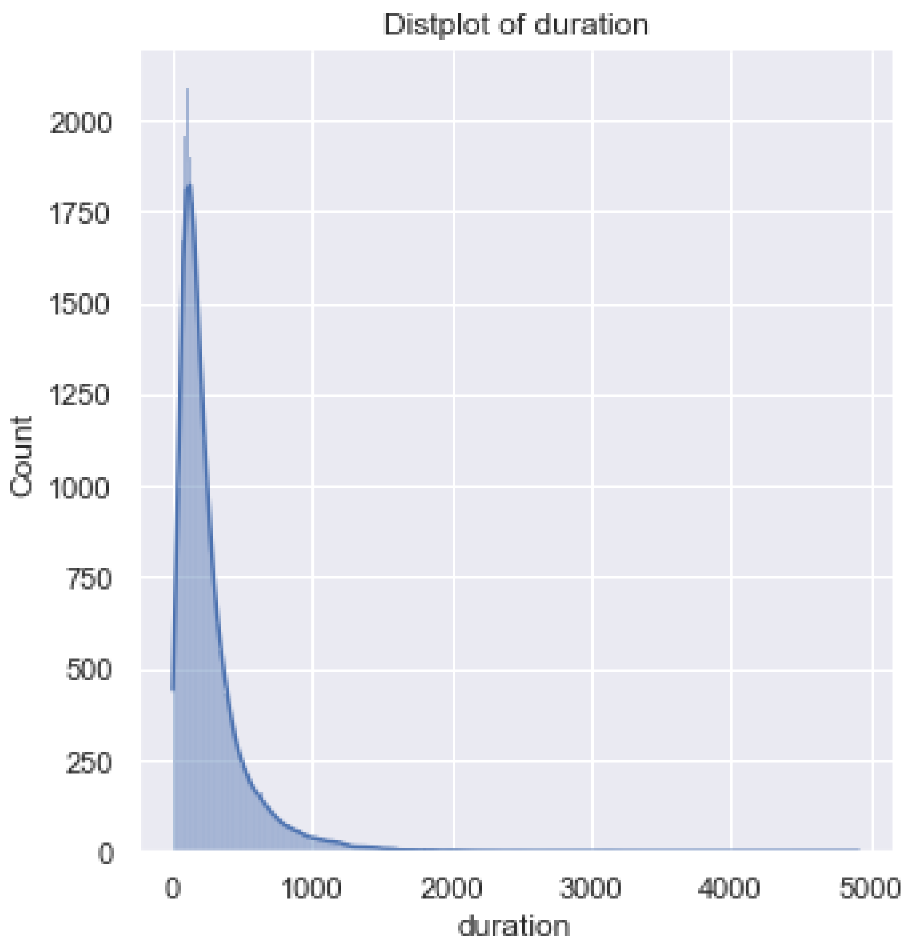

Countplot of education
25000
20000
15000
10000
5000
0
count
tertiary
secondary
primary
education

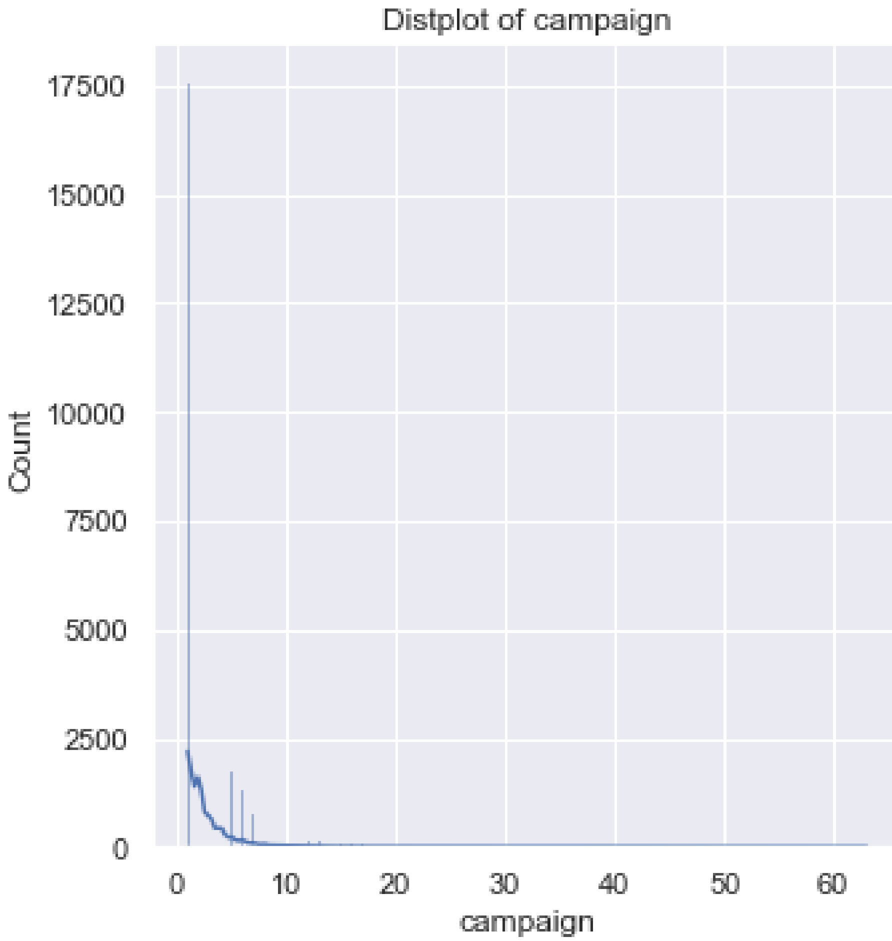

Distplot of balance
7000
6000
5000
4000
3000
2000
1000
0
Count
0
20000
40000
60000
80000
100000
balance

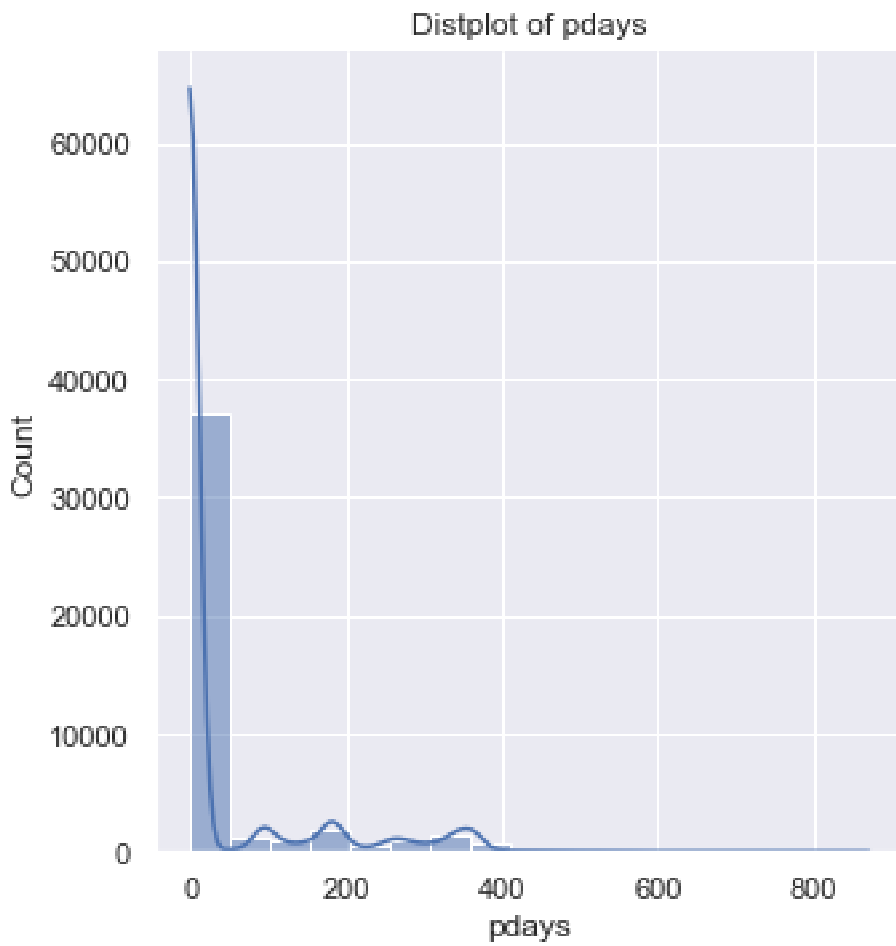

Countplot of contact
30000
25000
20000
15000
10000
5000
0
count
telephone
cellular
contact

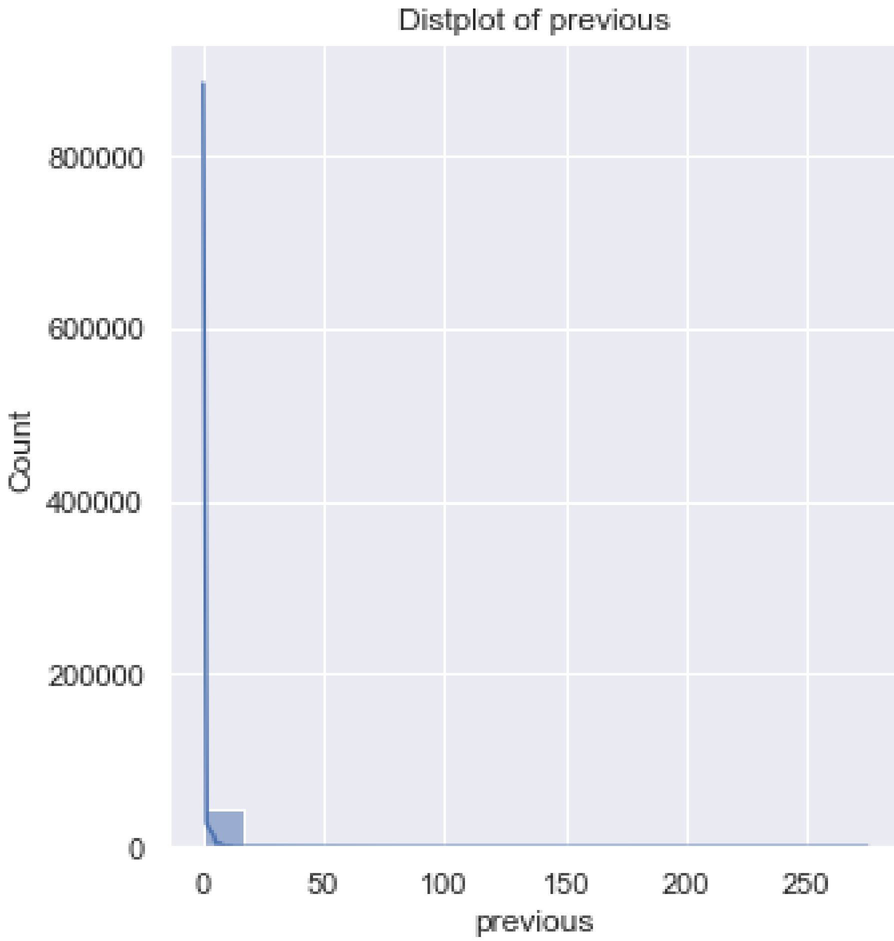

Distplot of day
Count
0
500
1000
1500
2000
2500
day
0
5
10
15
20
25
30

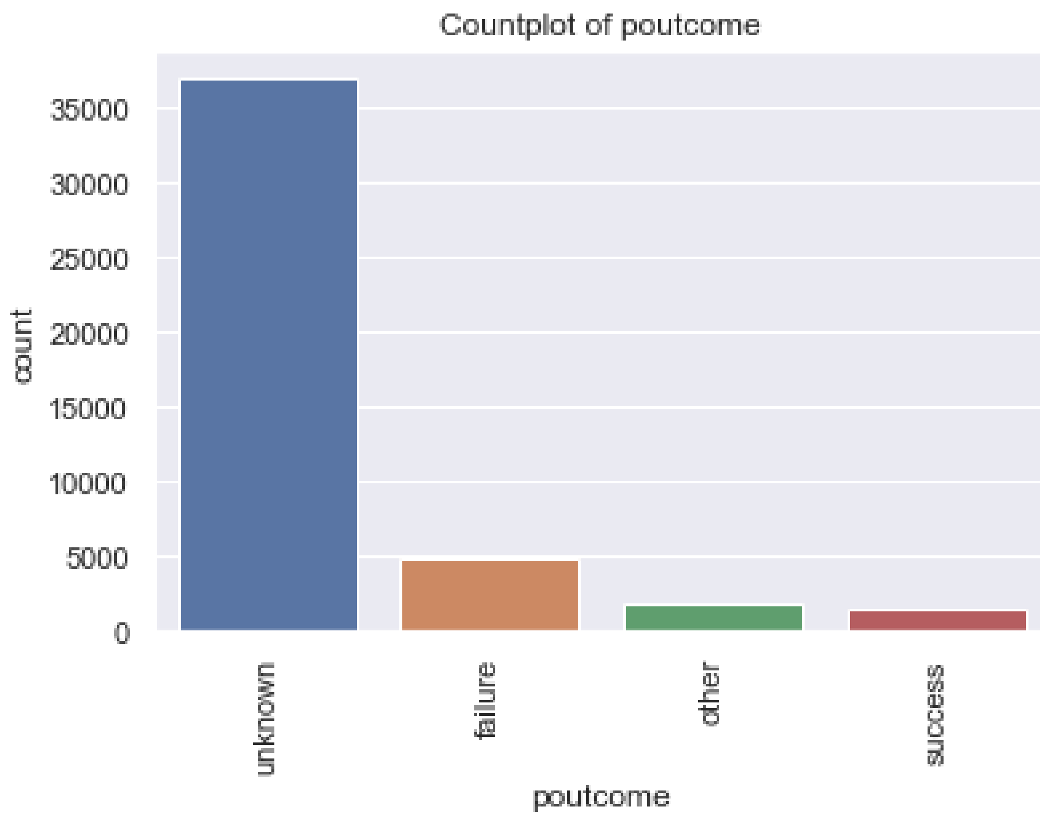

Countplot of month
14000
12000
10000
8000
6000
4000
2000
0
count
may
jun
jul
aug
oct
nov
dec
jan
feb
mar
apr
sep
month

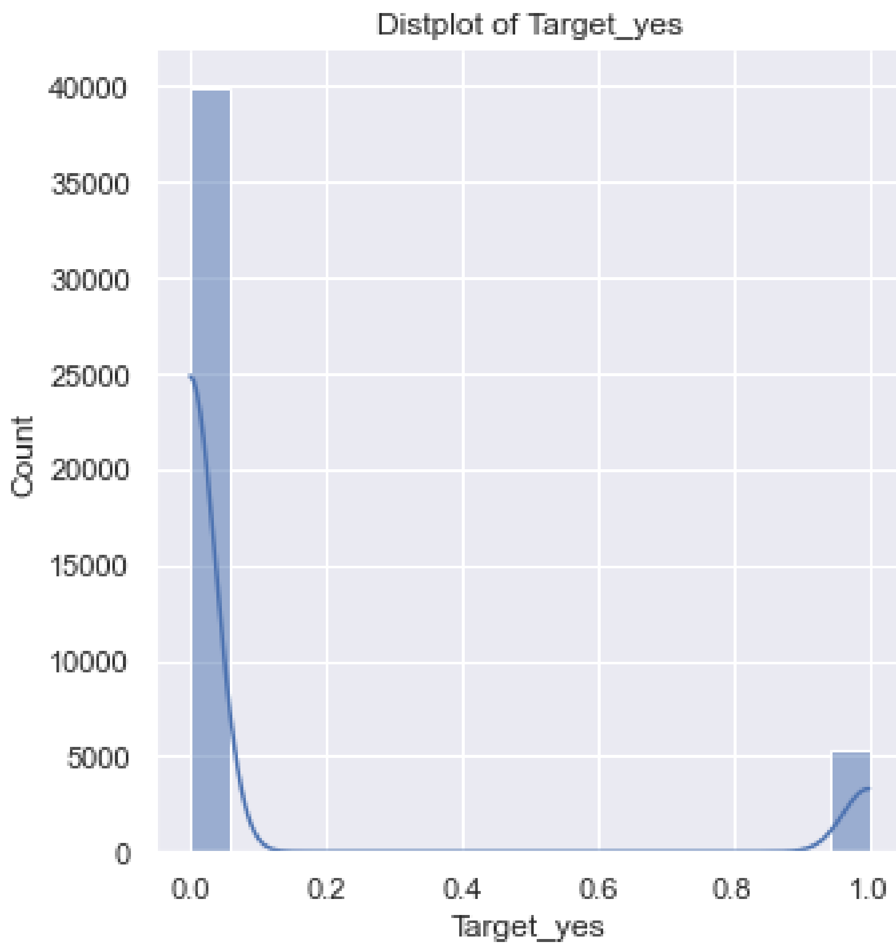

Distplot of duration
Count
2000
1750
1500
1250
1000
750
500
250
0
0
1000
2000
3000
4000
5000
duration

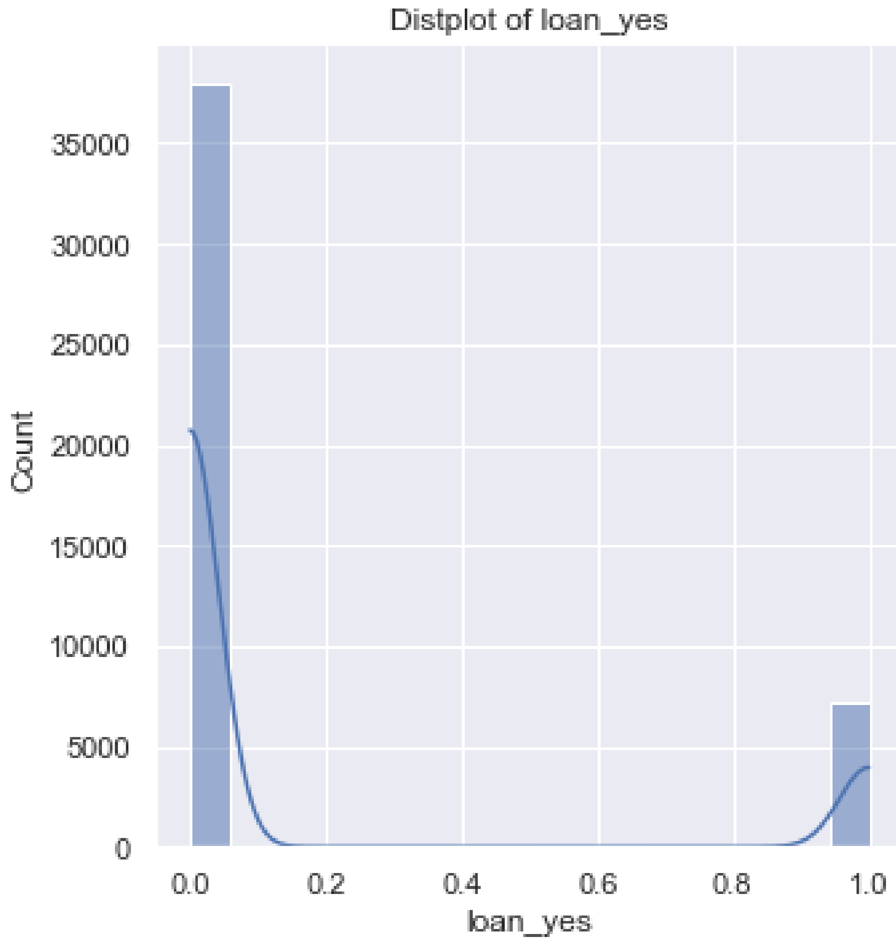

Distplot of campaign
17500
15000
12500
10000
7500
5000
2500
0
Count
0
10
20
30
40
50
60
campaign


Distplot of pdays
Count
60000
50000
40000
30000
20000
10000
0
0
200
400
600
800
pdays

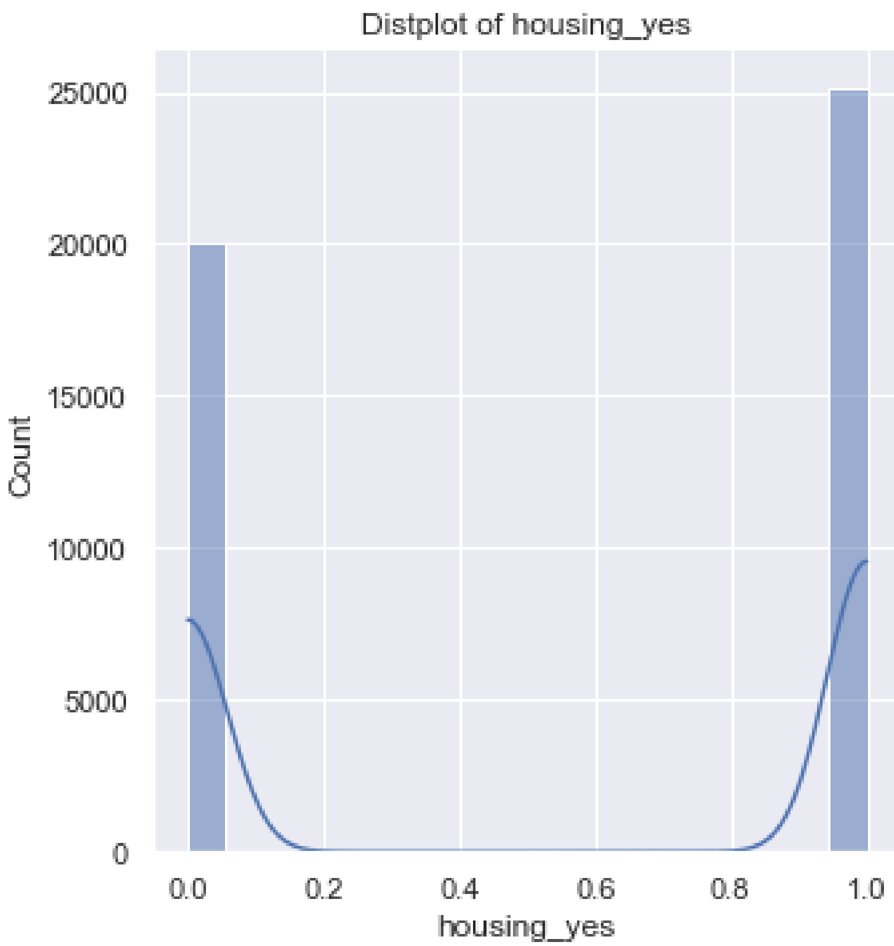

Distplot of previous
800000
600000
400000
200000
0
Count
0
50
100
150
200
250
previous

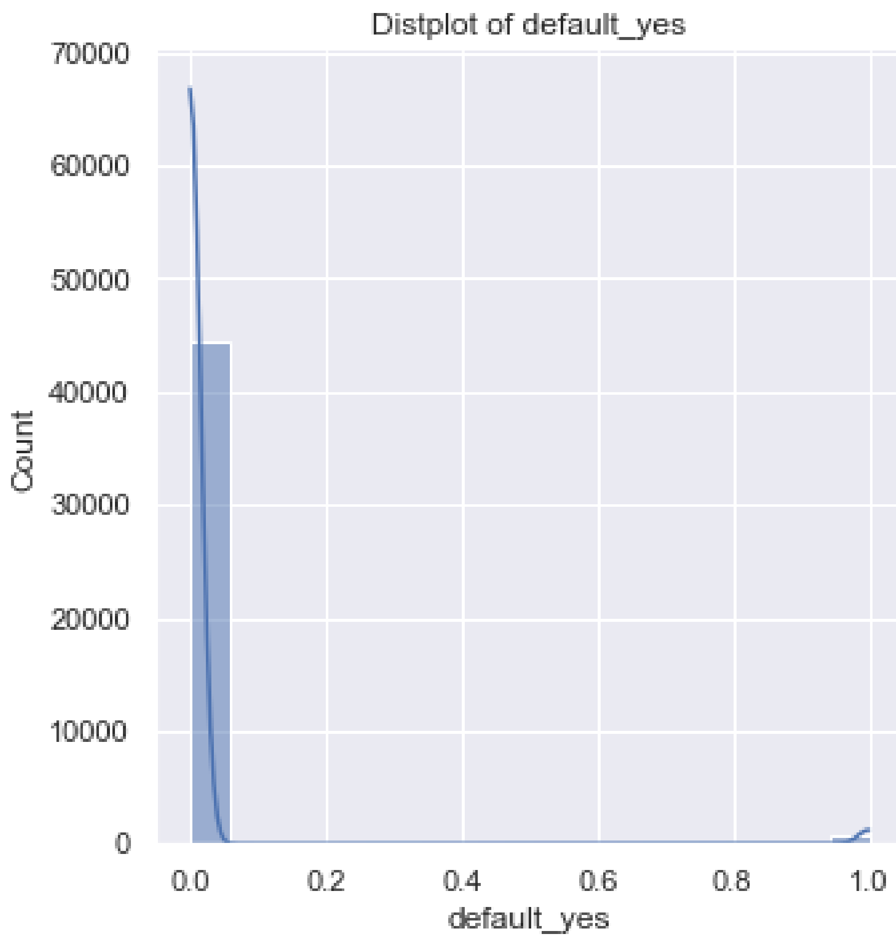

Countplot of poutcome
35000
30000
25000
20000
15000
10000
5000
0
count
unknown
failure
other
success
poutcome

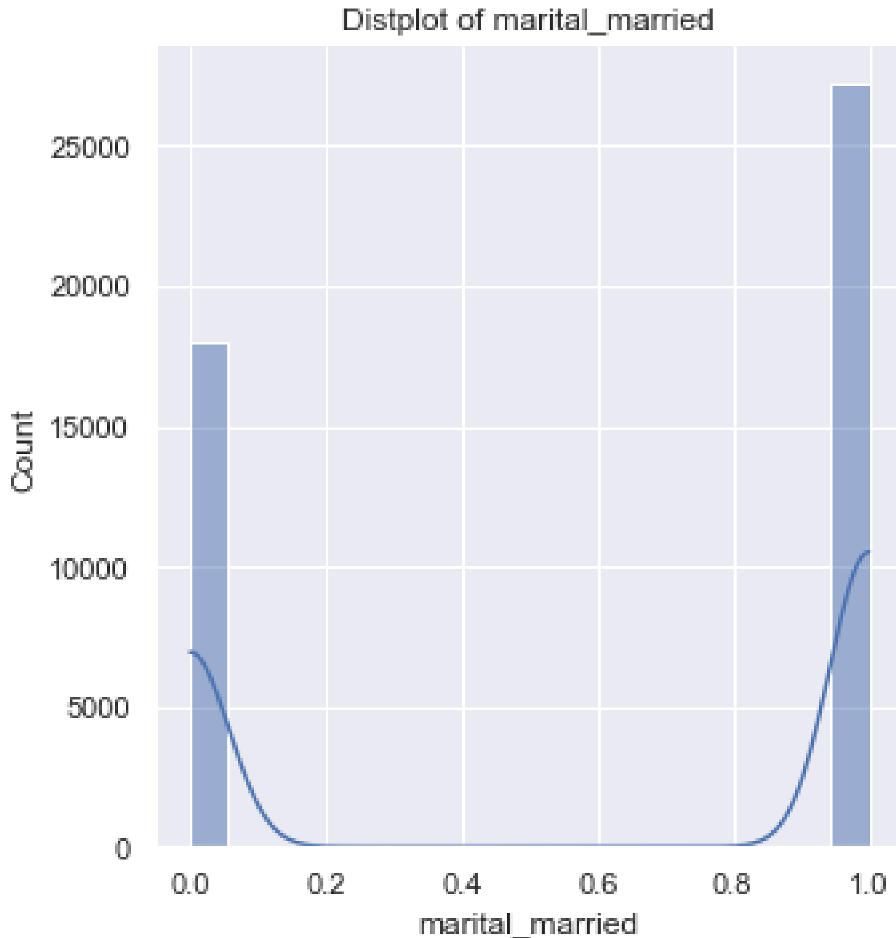

Distplot of Target_yes
Count
40000
35000
30000
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0
Target_yes



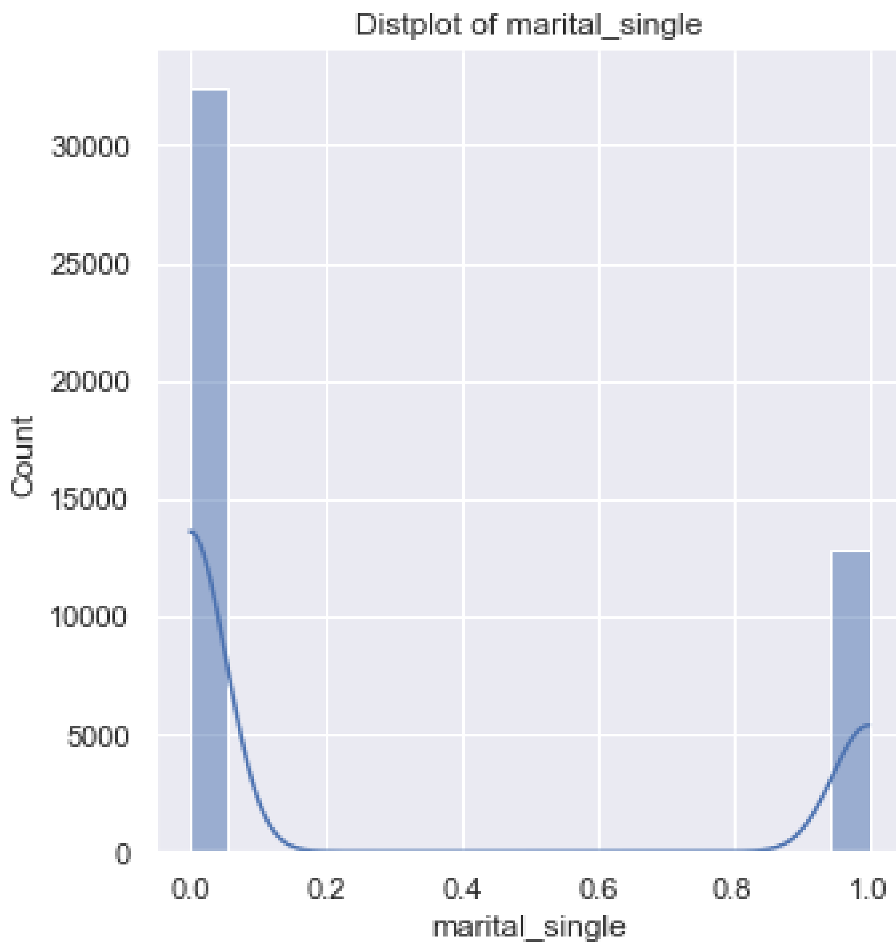
Distplot of loan_yes
Count
35000
30000
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0
loan_yes


Distplot of housing_yes
Count
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0
housing_yes


Distplot of default_yes
70000
60000
50000
40000
30000
20000
10000
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
default_yes



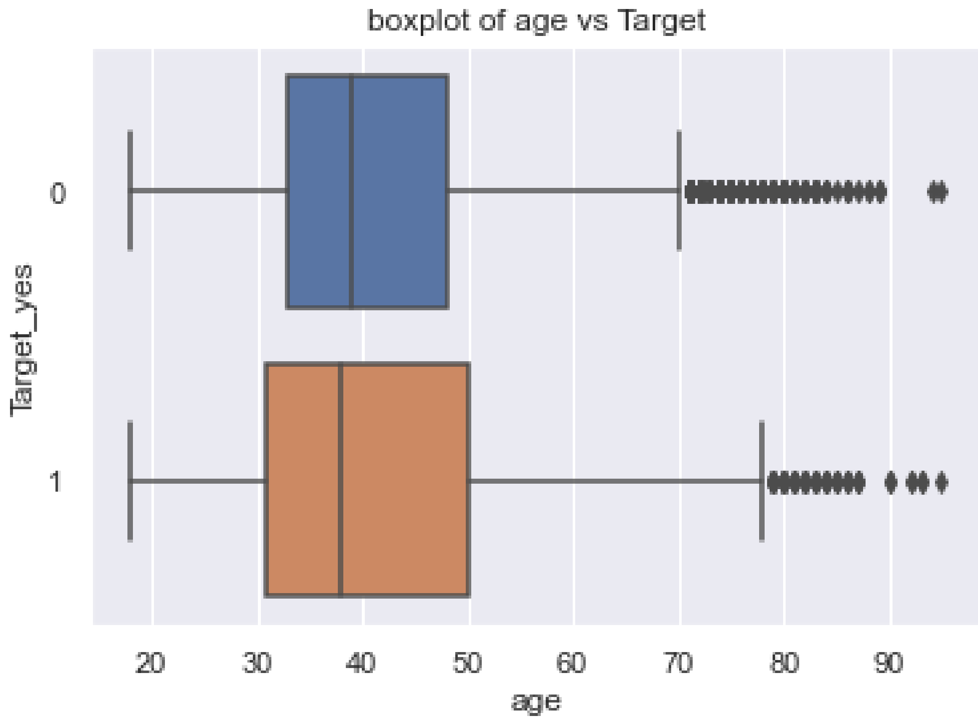
Distplot of marital_married
Count
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0
marital_married

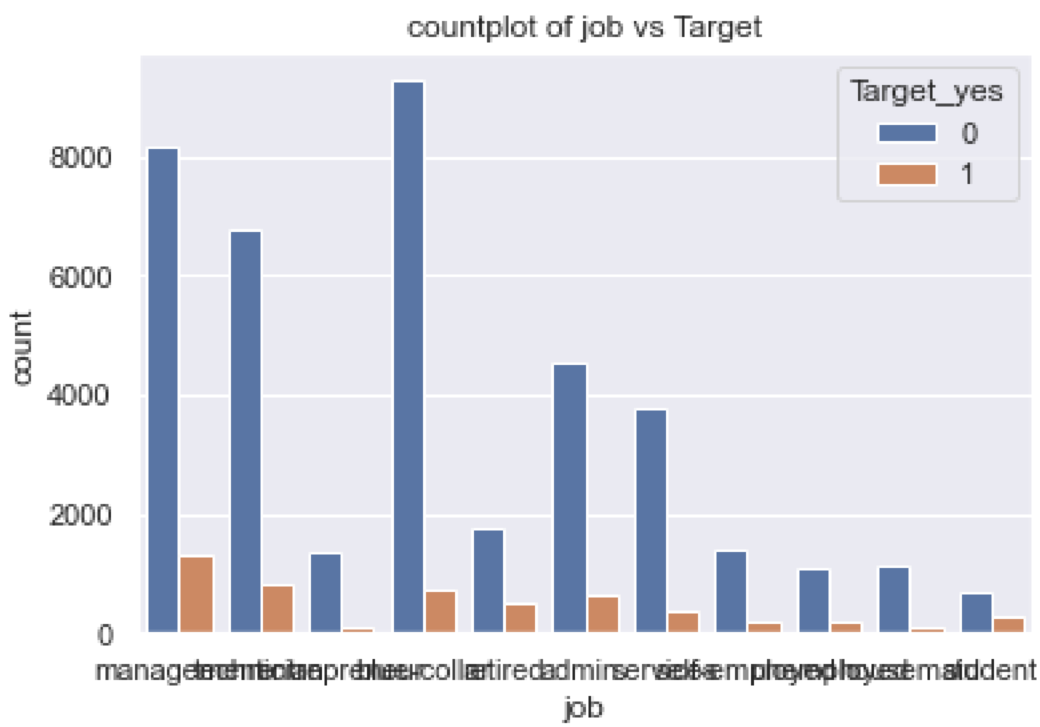


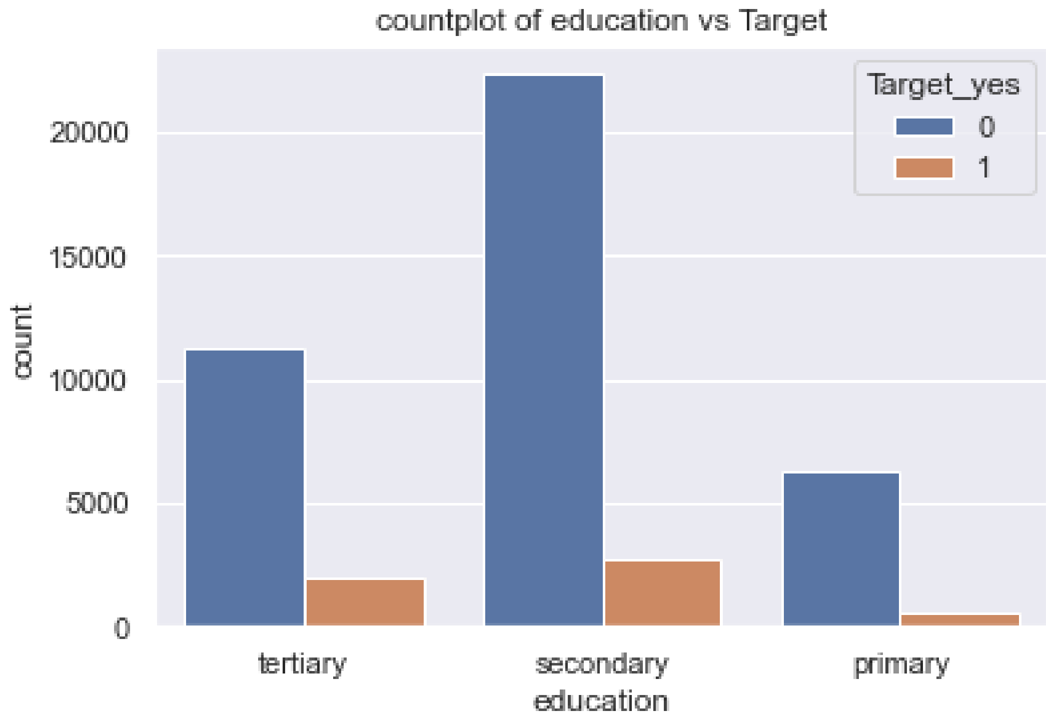
3.0.2 Multivariate analysis (6marks)

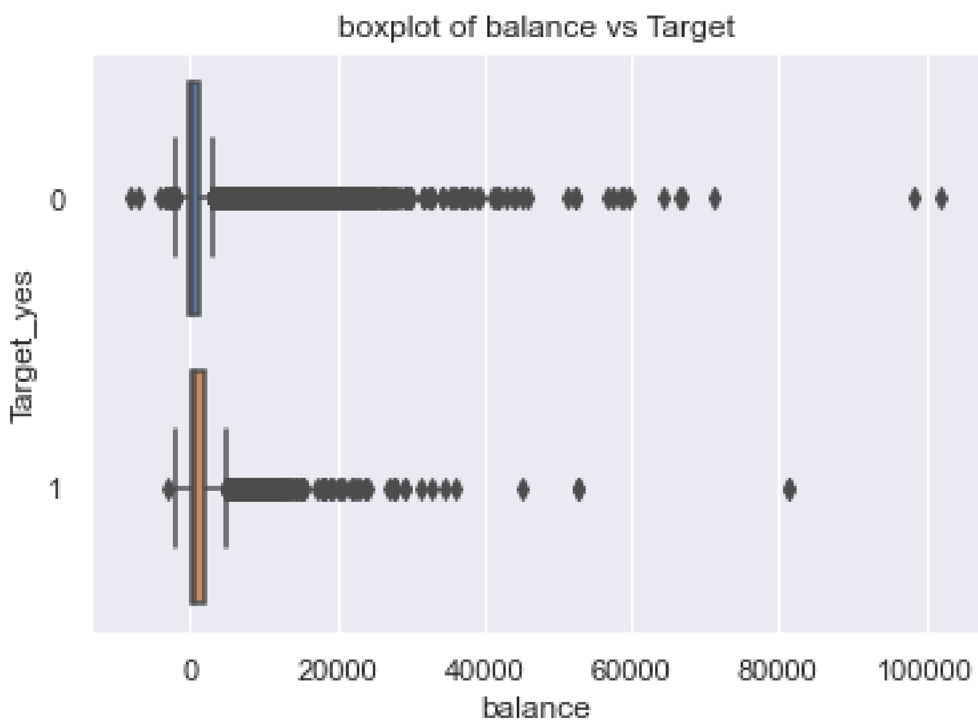
Make a function to plot boxplots for all continuous variables VS 'Target' variable and countplots for all categorical variables VS 'Target' variable? - 3 Marks

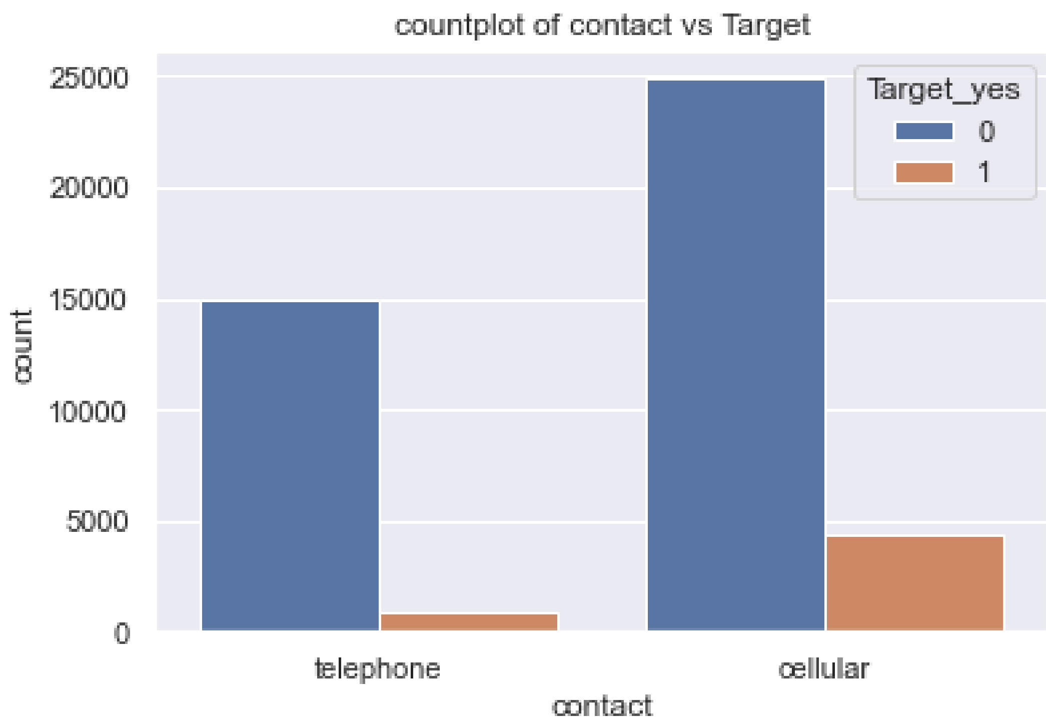
[10]:
```
for col in df.columns:
    if df[col].dtypes == 'object':
        sns.countplot(data=df, x=col, hue='Target_yes')
        temp = "countplot of "+col+' vs Target'
        plt.title(temp)
        plt.show()
    else:
        sns.boxplot(data=df, x=col, y='Target_yes', orient='h')
        temp = 'boxplot of '+ col + ' vs Target'
        plt.title(temp)
        plt.show()
```

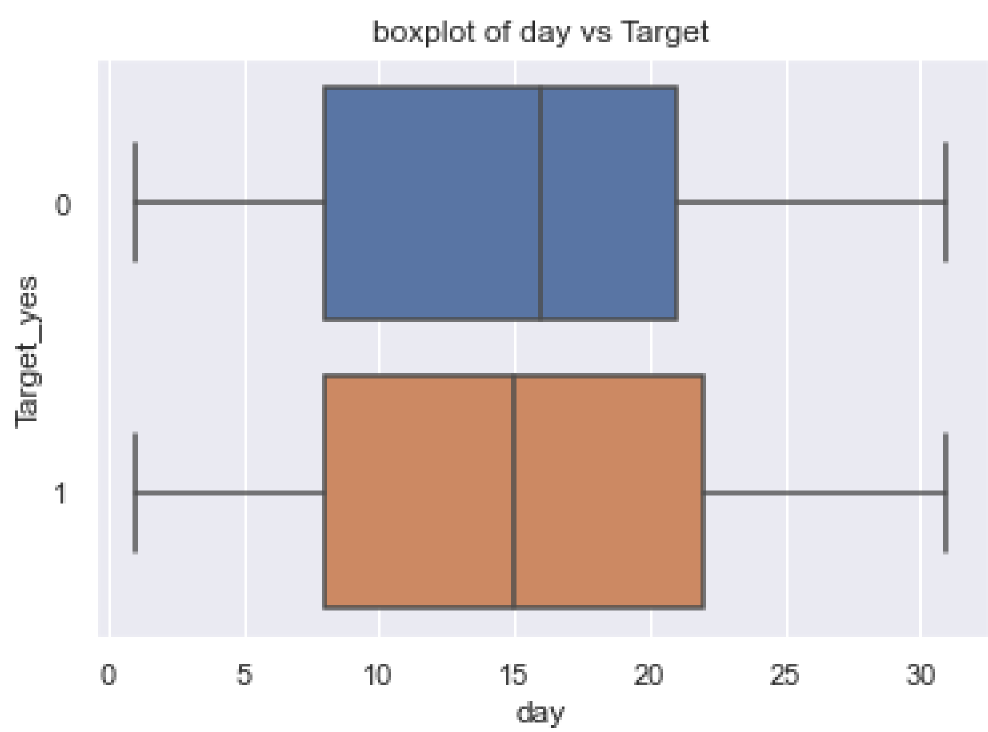

boxplot of age vs Target
Target_yes
0
1
20
30
40
50
60
70
80
90
age

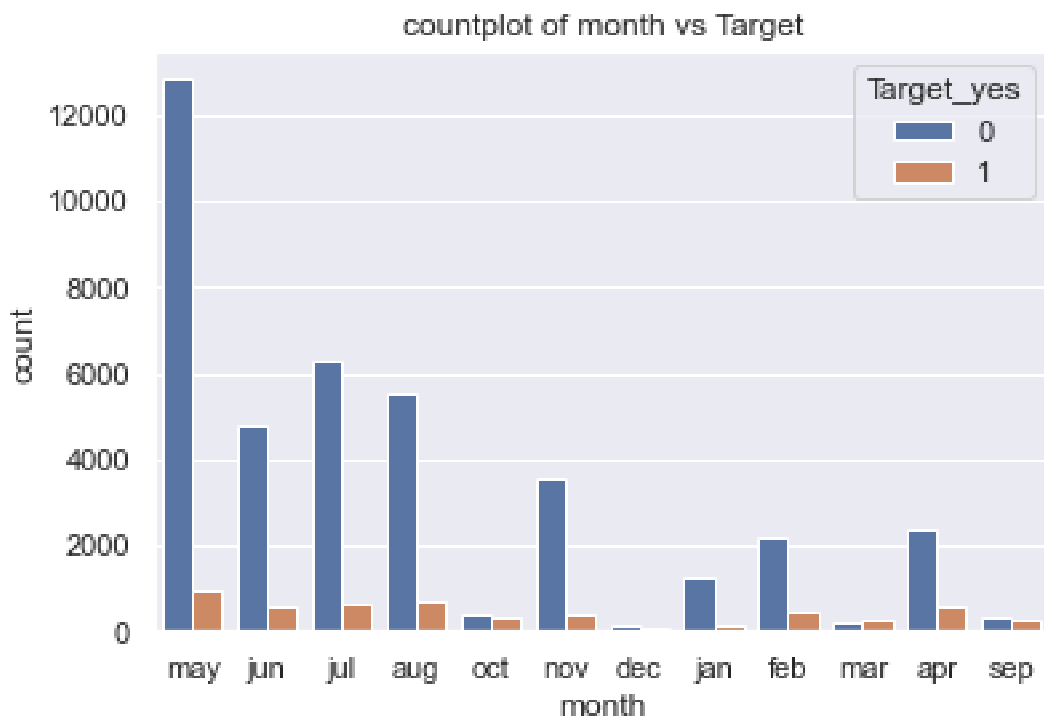

countplot of job vs Target
Target_yes
0
1
count
8000
6000
4000
2000
0
job

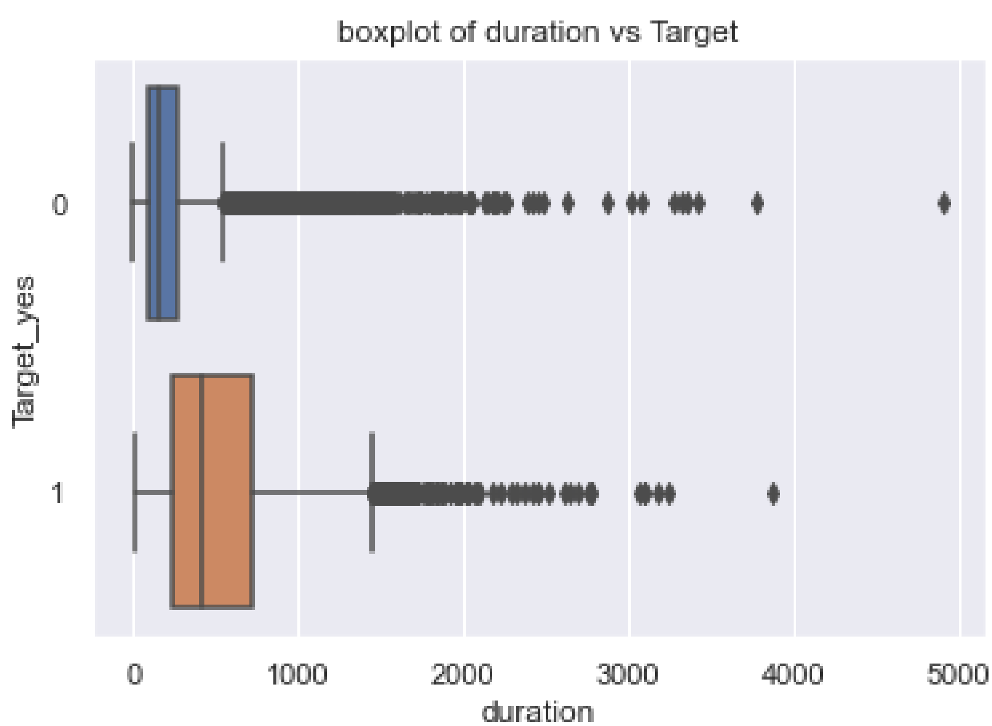

countplot of education vs Target
Target_yes
0
1
20000
15000
10000
5000
0
count
tertiary
secondary
primary
education

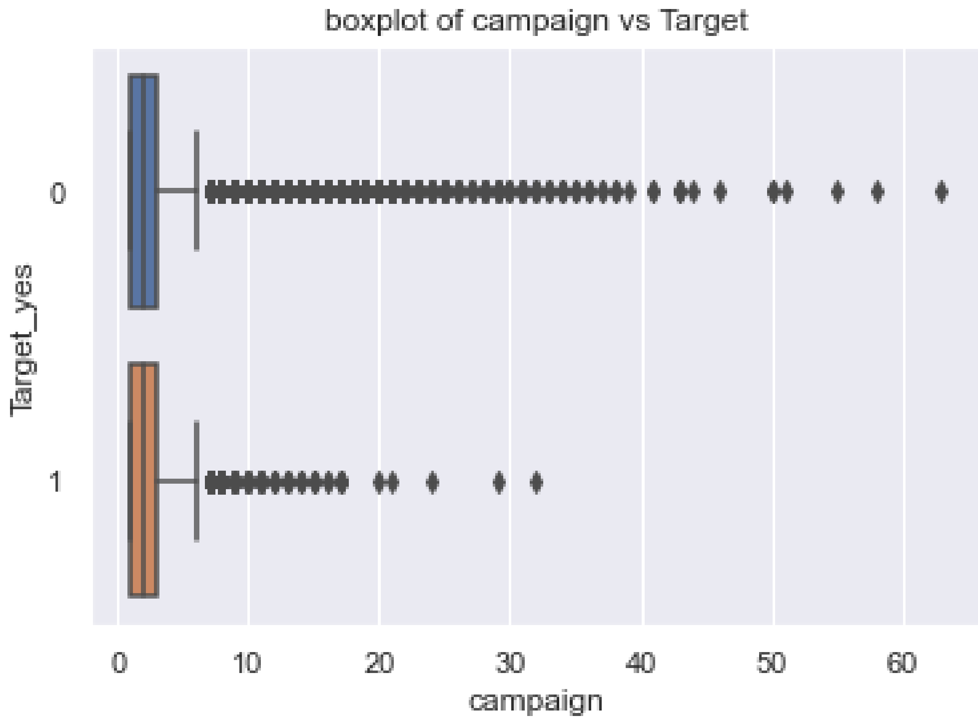

boxplot of balance vs Target
0
1
Target_yes
0
20000
40000
60000
80000
100000
balance

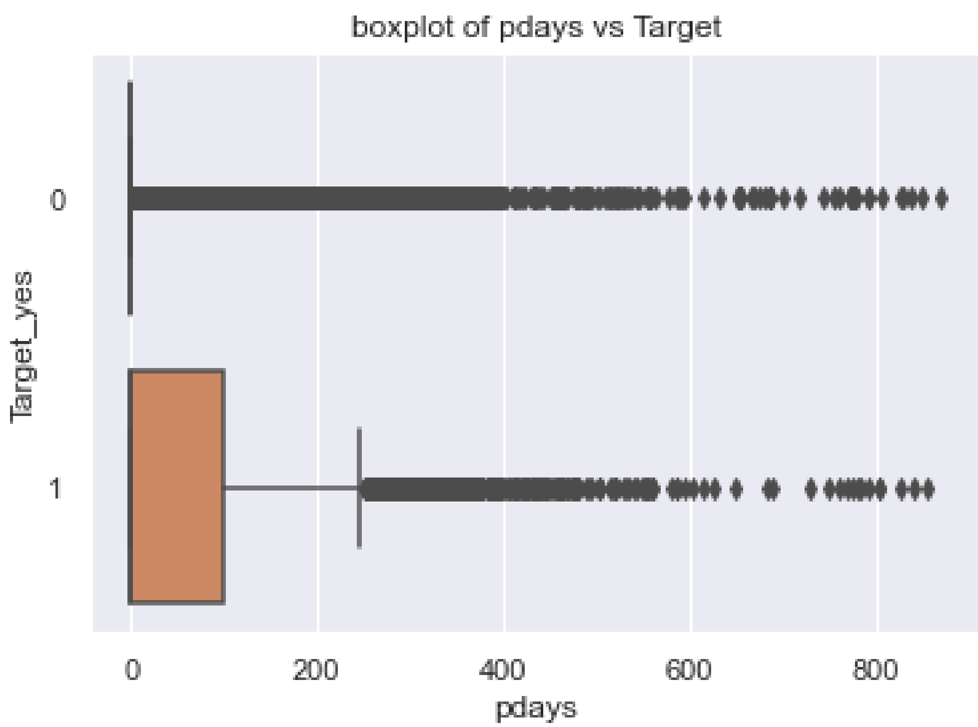

countplot of contact vs Target
Target_yes
0
1
25000
20000
15000
10000
5000
0
count
telephone
cellular
contact

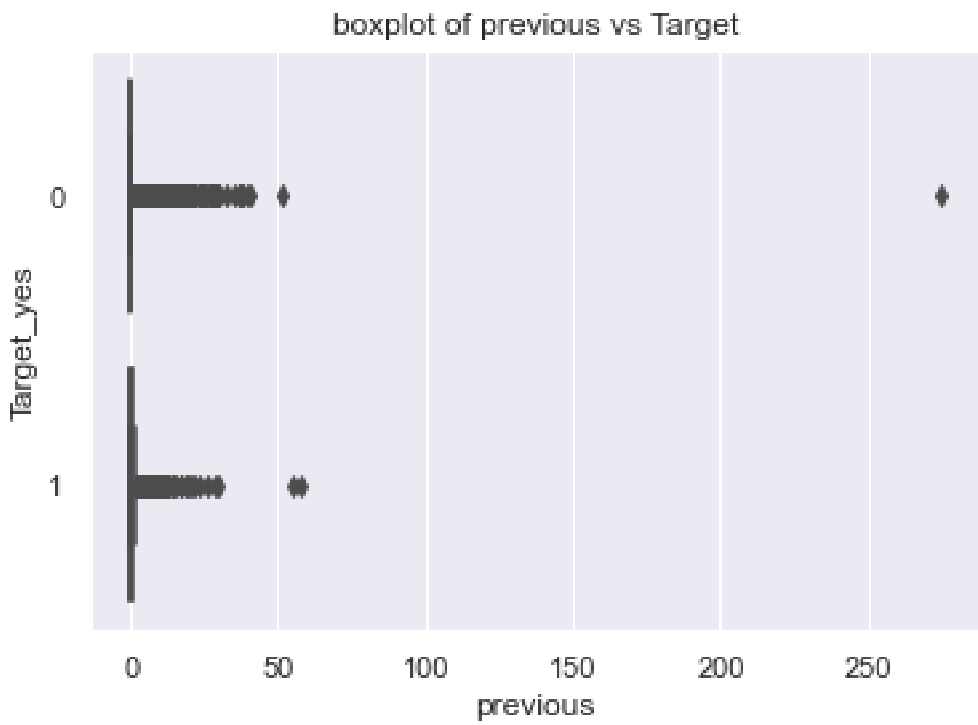

boxplot of day vs Target
0
1
Target_yes
0
5
10
15
20
25
30
day

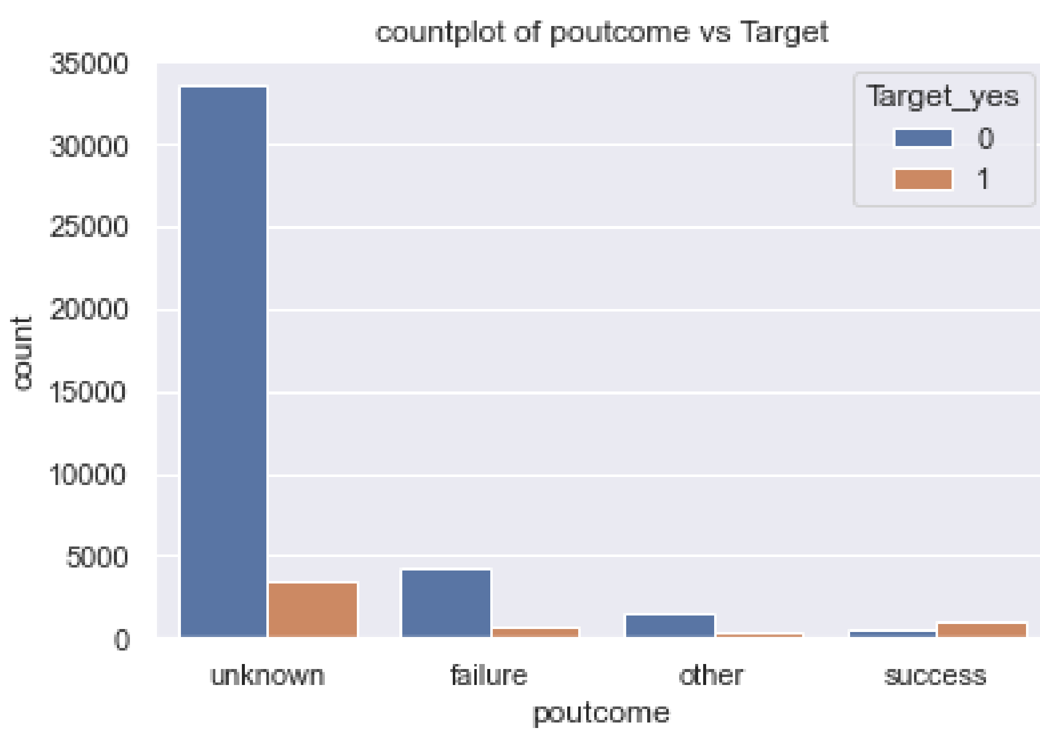

countplot of month vs Target
Target_yes
0
1
12000
10000
8000
6000
4000
2000
0
count
may
jun
jul
aug
oct
nov
dec
jan
feb
mar
apr
sep
month

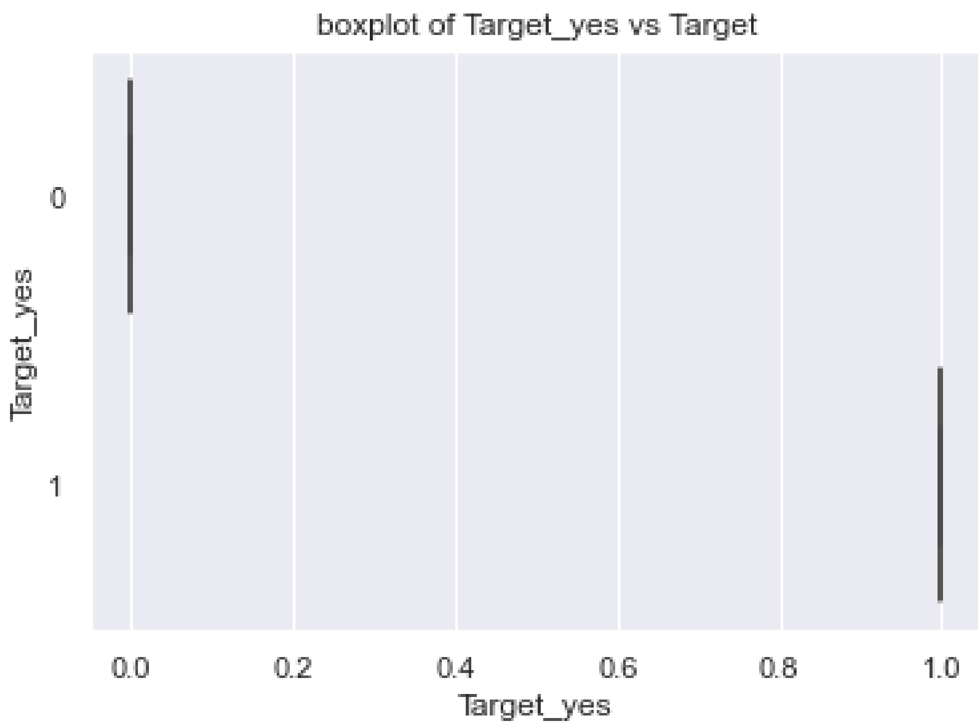

boxplot of duration vs Target
0
1
Target_yes
0
1000
2000
3000
4000
5000
duration


boxplot of campaign vs Target
Target_yes
0
1
0
10
20
30
40
50
60
campaign

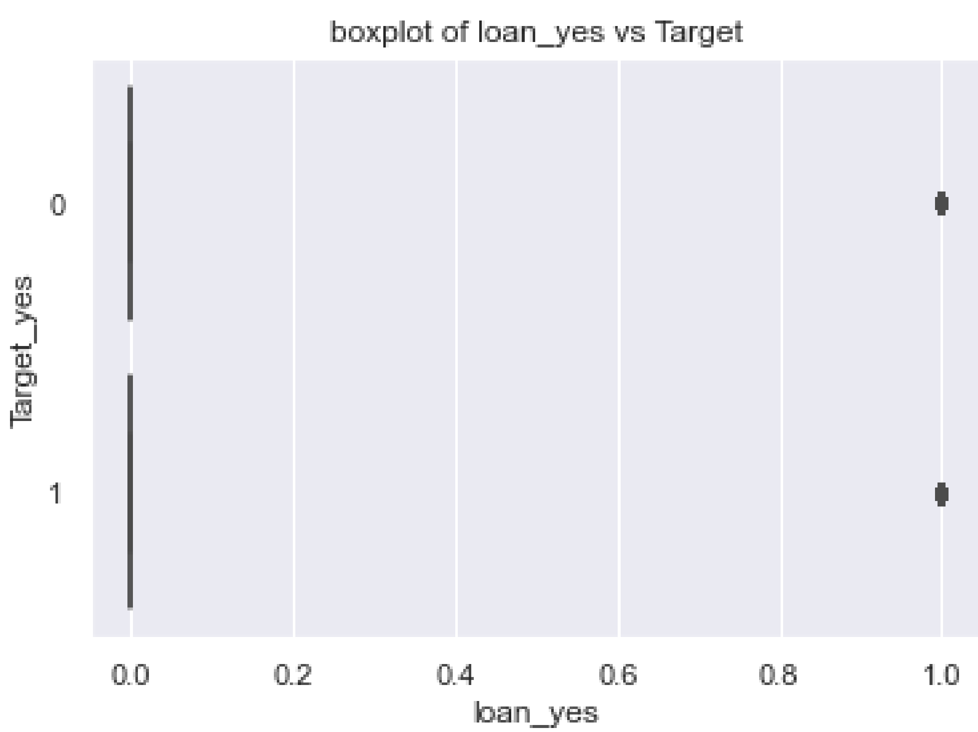

boxplot of pdays vs Target
Target_yes
0
1
0
200
400
600
800
pdays


boxplot of previous vs Target
Target_yes
0
1
0
50
100
150
200
250
previous

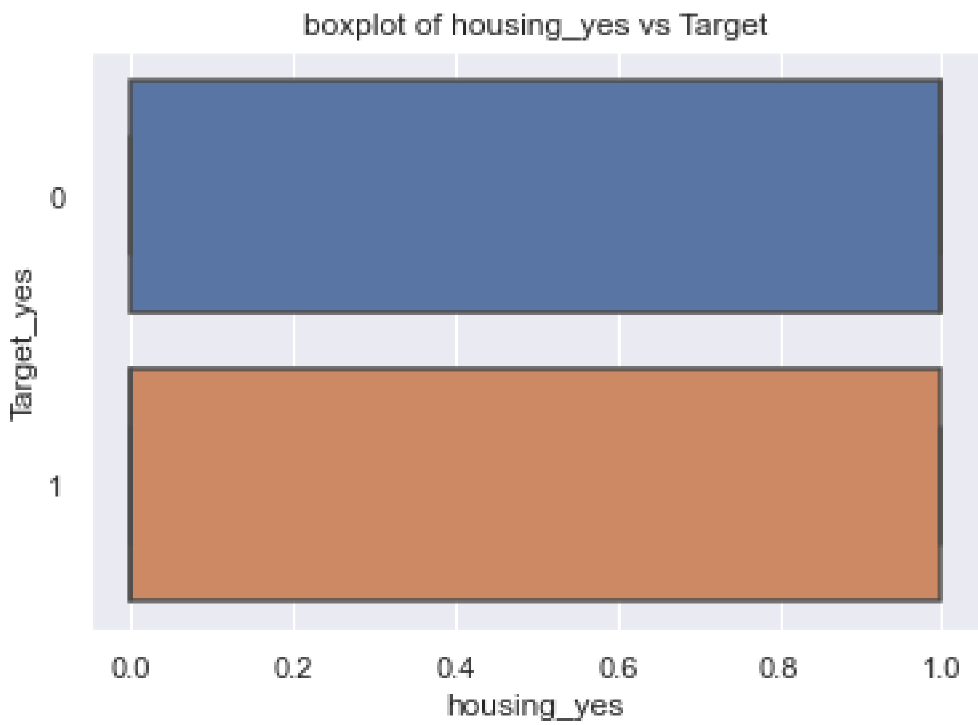

countplot of poutcome vs Target
count
35000
30000
25000
20000
15000
10000
5000
0
Target_yes
0
1
unknown
failure
other
success
poutcome

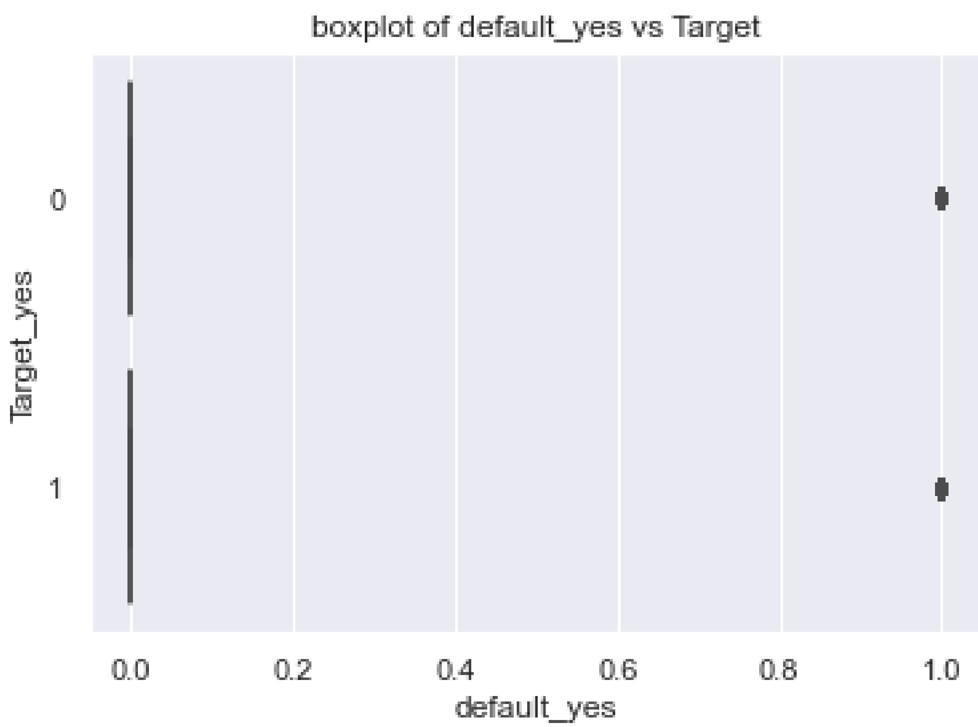


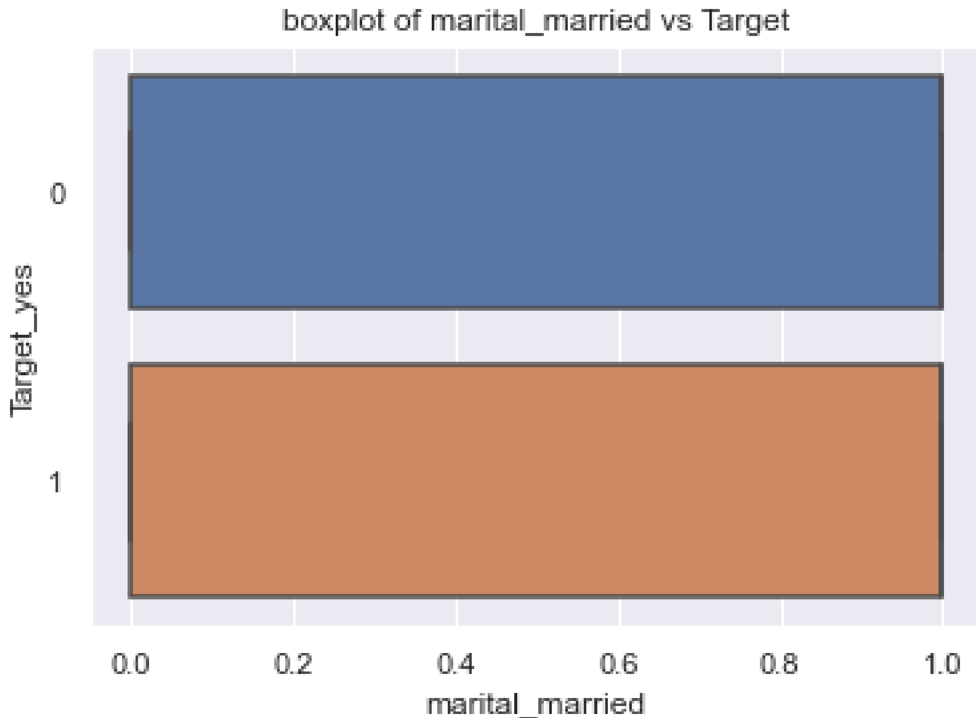
boxplot of Target_yes vs Target
Target_yes
0
1
0.0
0.2
0.4
0.6
0.8
1.0
Target_yes

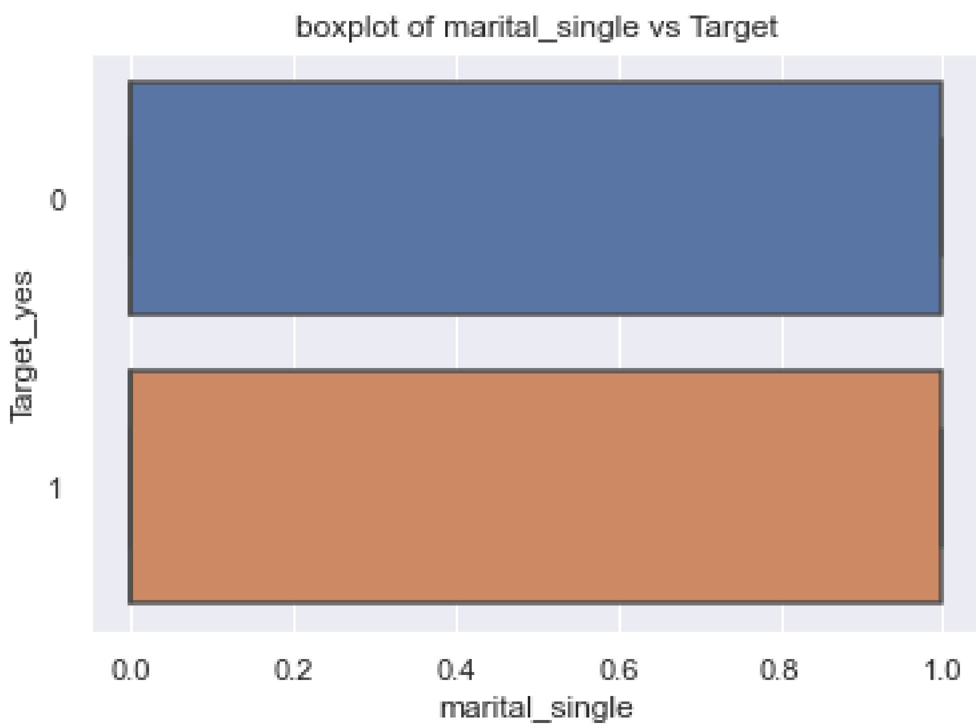


boxplot of loan_yes vs Target
Target_yes
0
1
0.0
0.2
0.4
0.6
0.8
1.0
loan_yes


boxplot of housing_yes vs Target
Target_yes
0
1
0.0
0.2
0.4
0.6
0.8
1.0
housing_yes

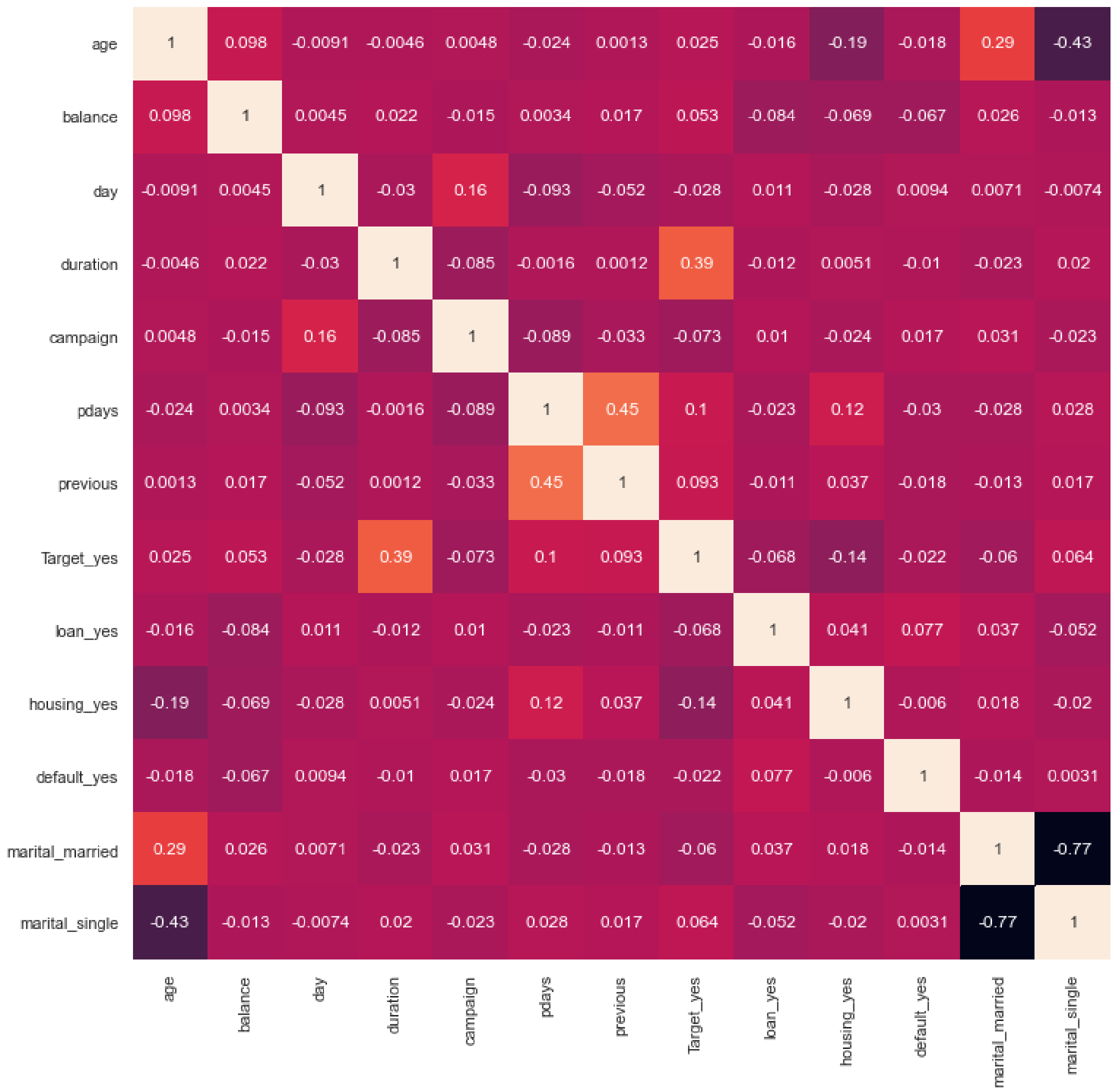

boxplot of default_yes vs Target
Target_yes
0
1
0.0
0.2
0.4
0.6
0.8
1.0
default_yes


boxplot of marital_married vs Target
0
1
Target_yes
0.0
0.2
0.4
0.6
0.8
1.0
marital_married

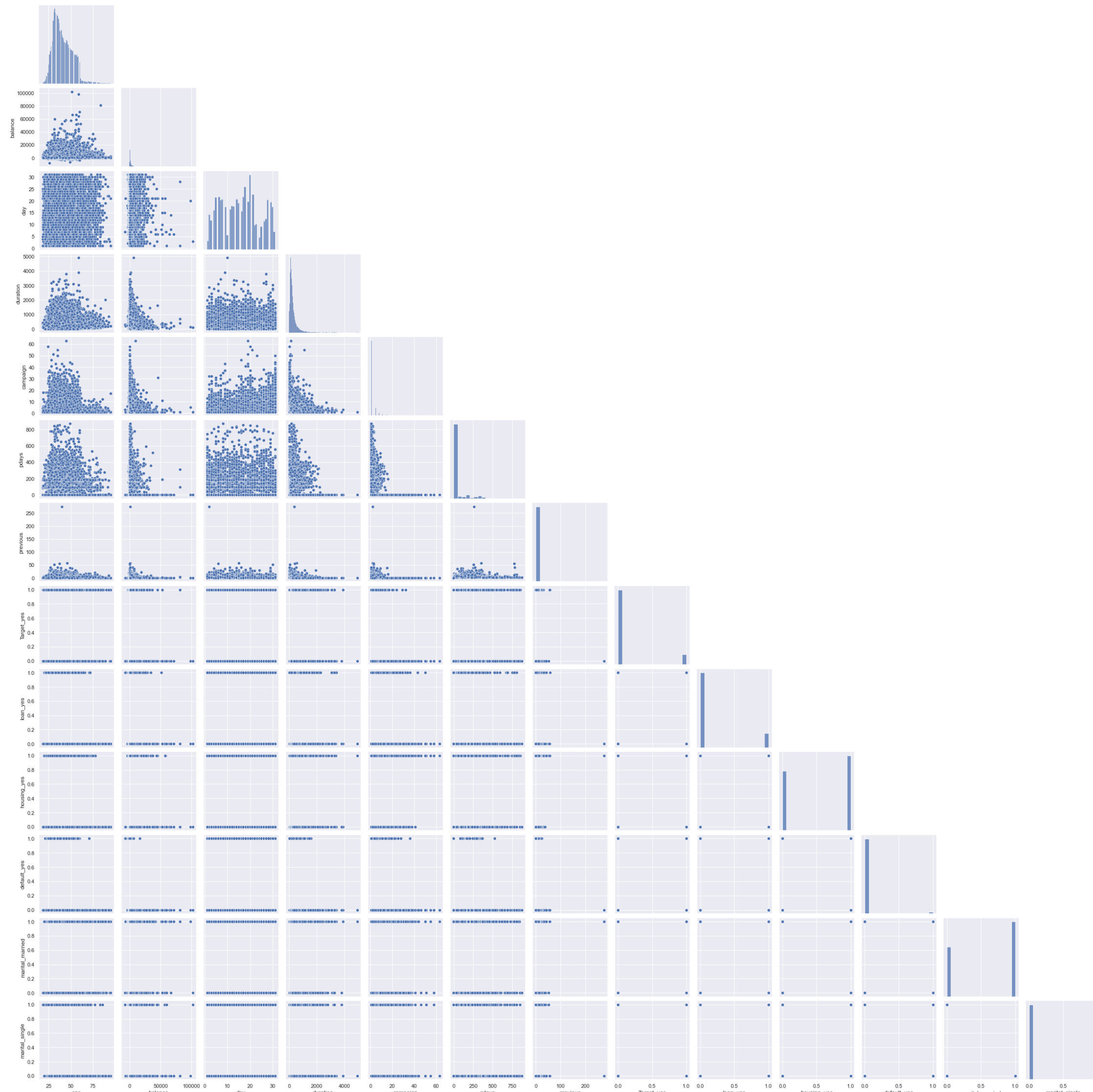

boxplot of marital_single vs Target
0
1
Target_yes
0.0
0.2
0.4
0.6
0.8
1.0
marital_single

Bi-variate analysis between predictor variables and target column. Comment on your findings in terms of their relationship and degree of relation if any. Visualize the analysis using pair plots, heatmaps, histograms or density curves. - 3 Marks

[11]:
```
plt.figure(figsize=(12,12));
sns.heatmap(df.corr(), cbar=False, annot=True);
```



[12]:
```
sns.pairplot(df, corner=True);
```



insights:

There is a strong correlation between the duration spent speaking to the client and the client purchasing the offering. There also seems to be little or no value contacting clients over 35 times, if resources can be better spent increasing durations of other calls.

4 Deliverable –2 (Prepare the data for analytics)–(5)

Label encode or create dummy variables for categorical variables. Give reason for selecting either of them. - 2 Marks

[13]:
```
#using dummies for all remaining categorical variables
cols = []
```

```
for col in df.columns:
    if df[col].dtypes == "object":
        cols.append(col)
df = pd.get_dummies(df, columns=cols, drop_first=True)
df.head()
```

[13]:
```
   age  balance  day  duration  campaign  pdays  previous  Target_yes  \
0   58     2143    5       261         1     -1         0           0
1   44       29    5       151         1     -1         0           0
2   33        2    5        76         1     -1         0           0
3   47     1506    5        92         1     -1         0           0
4   33        1    5       198         1     -1         0           0

   loan_yes  housing_yes  …  month_jul  month_jun  month_mar  month_may  \
0         0            1  …          0          0          0          1
1         0            1  …          0          0          0          1
2         1            1  …          0          0          0          1
3         0            1  …          0          0          0          1
4         0            0  …          0          0          0          1

   month_nov  month_oct  month_sep  poutcome_other  poutcome_success  \
0          0          0          0               0                 0
1          0          0          0               0                 0
2          0          0          0               0                 0
3          0          0          0               0                 0
4          0          0          0               0                 0

   poutcome_unknown
0                 1
1                 1
2                 1
3                 1
4                 1

[5 rows x 40 columns]
```

[ ]:

Create the training set and test set in a ratio of 70:30. Make sure and verify distribution of classes is the same in the full dataset and train test split data. - 3 Marks

[14]:
```
y = df['Target_yes']
x = df.drop('Target_yes', axis=1)
```

[15]:
```
x_train, x_test, y_train, y_test = train_test_split(x,y,random_state=7,␣
↪test_size = 0.3)
```

[16]:
```
print('training data as a percentage of total data', round(len(x_train)/len(x)␣
↪*100,2))
print('testing data as a percentage of total data', round(len(x_test)/len(x)␣
↪*100,2))

print('\n\ntraining results as a percentage of total data', round(len(y_train)/
↪len(y) *100,2))
print('testing results as a percentage of total data', round(len(y_test)/len(y)␣
↪*100,2))
```

```
training data as a percentage of total data 70.0
testing data as a percentage of total data 30.0


training results as a percentage of total data 70.0
testing results as a percentage of total data 30.0
```

5 Deliverable –3 (Create the ensemble model)–(30)

Build the ensemble models (Bagging and Boosting) and Decision Tree model (at least 4 models in total). Note the model performance by using different metrics. Use confusion matrix to evaluate class level metrics i.e. Precision/Recall. - 10 Marks

[17]:
```
precision_test,recall_test, accuracy_test, roc_auc_test,f1_test = [],[],[],[],[]
precision_train,recall_train, accuracy_train, roc_auc_train,f1_train =␣
↪[],[],[],[],[]
precision_xval,recall_xval, accuracy_xval, roc_auc_xval,f1_xval = [],[],[],[],[]
columns = ['Testing Precision', 'Testing Recall', 'Testing Accuracy','Testing␣
↪ROC_AUC', 'Testing F1', \
           'Training Precision', 'Training Recall', 'Training␣
↪Accuracy','Training ROC_AUC','Training F1',\
           'CrossValidation Precision', 'CrossValidation Recall',␣
↪'CrossValidation Accuracy','CrossValidation ROC_AUC', 'CrossValidation F1']
models = ['Decision Tree', 'Bagging','Random Forest', 'Adaboost',␣
↪'Gradientboost']
def printReports(y_test,x_test, y_train, x_train,  model):
    print(classification_report(y_test, y_pred), '\n')
    sns.heatmap(confusion_matrix(y_test, y_pred), cbar = False, annot = True,␣
↪fmt='.5g')
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()

    #load arrays with training performance data
    precision_train.append(precision_score(y_train, model.predict(x_train)))
    recall_train.append(recall_score(y_train, model.predict(x_train)))
    accuracy_train.append(accuracy_score(y_train, model.predict(x_train)))
```

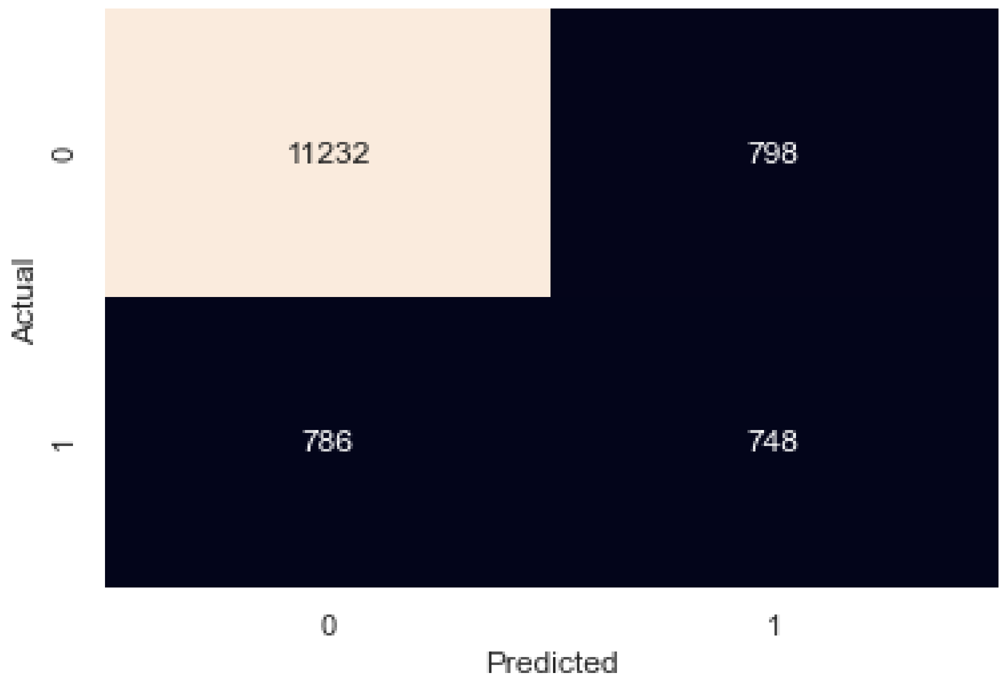
```
    roc_auc_train.append(roc_auc_score(y_train, model.predict(x_train)))
    f1_train.append(f1_score(y_train, model.predict(x_train)))
    print('***TRAINING DATA***')
    print('precision score: ', precision_score(y_train, model.predict(x_train)))
    print('recall score: ', recall_score(y_train, model.predict(x_train)))
    print('accuracy score: ', accuracy_score(y_train, model.predict(x_train)))
    print('ROC AUC score: ', roc_auc_score(y_train, model.predict(x_train)))
    print('F1 score: ', f1_score(y_train, model.predict(x_train)))


    #load arrays with testing performance data
    precision_test.append(precision_score(y_test, model.predict(x_test)))
    recall_test.append(recall_score(y_test, model.predict(x_test)))
    accuracy_test.append(accuracy_score(y_test, model.predict(x_test)))
    roc_auc_test.append(roc_auc_score(y_test, model.predict(x_test)))
    f1_test.append(f1_score(y_test, model.predict(x_test)))
    print('\n\n***TESTING DATA***')
    print('precision score: ', precision_score(y_test, model.predict(x_test)))
    print('recall score: ', recall_score(y_test, model.predict(x_test)))
    print('accuracy score: ', accuracy_score(y_test, model.predict(x_test)))
    print('ROC AUC score: ', roc_auc_score(y_test, model.predict(x_test)))
    print('F1 score: ', f1_score(y_test, model.predict(x_test)))

    #cross validation data
    print('\n\n***CROSS VALIDATION DATA***')
    array = [precision_xval,recall_xval, accuracy_xval, roc_auc_xval,f1_xval]
    scoring_metrics=['precision','recall', 'accuracy','roc_auc','f1']
    for i in range(len(scoring_metrics)):
        score=cross_val_score(model, x_test, y_test, cv=10,␣
↪scoring=scoring_metrics[i])
        print('%s score  %-.2f'%(scoring_metrics[i],score.mean()))
        array[i].append(score.mean())

def featureImportance(model):
    output = pd.DataFrame()
    feature_weights = model.feature_importances_
    output['feature_rating'] =  feature_weights[0:]
    output['features'] = x_train.columns
    print(output.sort_values(by=['feature_rating'], ascending = False,␣
↪ignore_index=True))
```

5.1 Decision Tree

[18]:
```
model = DecisionTreeClassifier(criterion='entropy', max_depth=20)
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
```

[19]:
```
printReports(y_test, x_test, y_train, x_train, model)
```

```
              precision    recall  f1-score   support

           0       0.93      0.93      0.93     12030
           1       0.48      0.49      0.49      1534

    accuracy                           0.88     13564
   macro avg       0.71      0.71      0.71     13564
weighted avg       0.88      0.88      0.88     13564
```




```
***TRAINING DATA***
precision score:  0.9200652528548124
recall score:  0.9011984021304926
accuracy score:  0.9789869497898694
ROC AUC score:  0.9453288726556666
F1 score:  0.9105341046683707


***TESTING DATA***
precision score:  0.48382923673997413
recall score:  0.4876140808344198
accuracy score:  0.8832202890002949
```

```
ROC AUC score:  0.7106399581229454
F1 score:  0.48571428571428577


***CROSS VALIDATION DATA***
precision score  0.46
recall score  0.45
accuracy score  0.88
roc_auc score  0.71
f1 score  0.45
```

[20]:

```
featureImportance(model)
```

	feature_rating	features
0	0.310157	duration
1	0.102614	poutcome_success
2	0.098382	balance
3	0.083118	age
4	0.079434	day
5	0.039471	pdays
6	0.038565	housing_yes
7	0.026398	campaign
8	0.021401	contact_telephone
9	0.016666	month_mar
10	0.015150	previous
11	0.013493	month_oct
12	0.012579	month_jun
13	0.010829	month_jul
14	0.010277	month_may
15	0.010046	month_nov
16	0.009697	job_management
17	0.008874	month_aug
18	0.008756	loan_yes
19	0.008674	marital_married
20	0.008189	month_feb
21	0.007909	education_tertiary
22	0.007122	job_technician
23	0.006564	month_jan
24	0.005804	marital_single
25	0.005564	education_secondary
26	0.004961	month_sep
27	0.004139	job_blue-collar
28	0.003736	job_student
29	0.003585	job_services
30	0.003499	job_unemployed
31	0.002577	job_housemaid
32	0.002246	job_retired
33	0.002214	default_yes

```
34        0.002029          month_dec
35        0.002028   job_self-employed
36        0.001758       poutcome_other
37        0.000919     job_entrepreneur
38        0.000575     poutcome_unknown
```

[ ]:

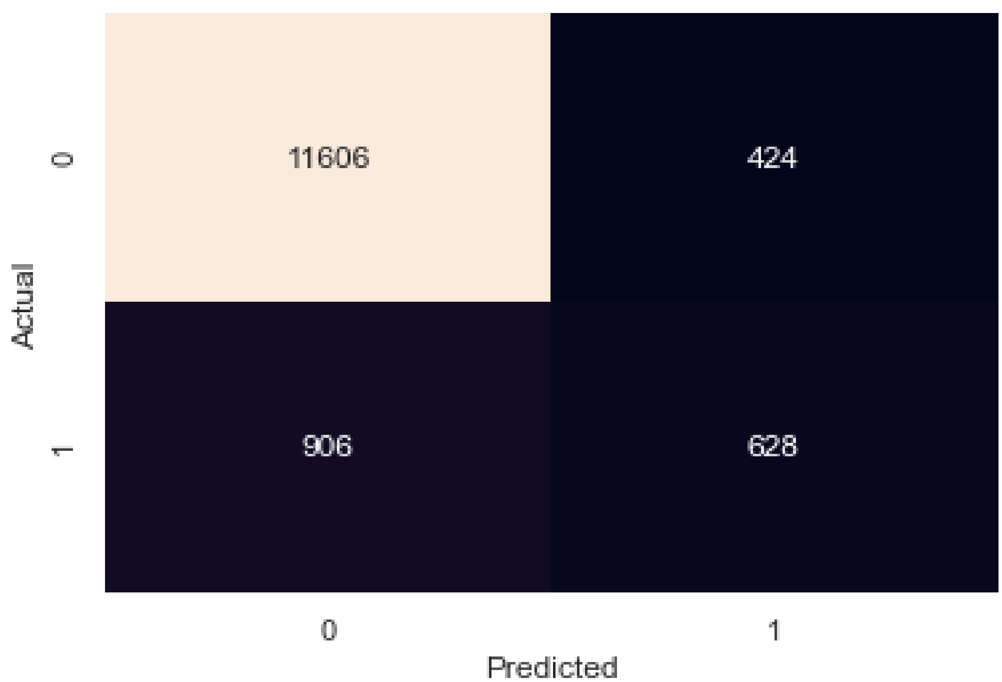
[ ]:

5.2 Bagging

[21]:
```python
model = BaggingClassifier()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
```

[22]:
```python
printReports(y_test, x_test, y_train, x_train, model)
```

```
              precision    recall  f1-score   support

           0       0.93      0.96      0.95     12030
           1       0.60      0.41      0.49      1534

    accuracy                           0.90     13564
   macro avg       0.76      0.69      0.72     13564
weighted avg       0.89      0.90      0.89     13564
```



```
***TRAINING DATA***
precision score:  0.9940644431882419
recall score:  0.9366178428761651
accuracy score:  0.9918159699181597
ROC AUC score:  0.9679324694088269
F1 score:  0.9644864938982586


***TESTING DATA***
precision score:  0.596958174904943
recall score:  0.409387222946545
accuracy score:  0.9019463285166618
ROC AUC score:  0.6870710013319591
F1 score:  0.48569218870843


***CROSS VALIDATION DATA***
precision score  0.58
recall score  0.38
accuracy score  0.90
roc_auc score  0.88
f1 score  0.46
```

[23]:
```
#featureImportance(model)
```

[24]:
```
#print('model.base_estimator_: ', model.base_estimator_)
#print('\nmodel.estimators_: ', model.estimators_)
#print('\nmodel.estimators_samples_: ', model.estimators_samples_)
#print('\nmodel.estimators_features_: ', model.estimators_features_)
```

Explain the confusion matrix related terms like recall, precision etc. Also, select the best metric to choose one of the models from above. Give your reason for the same. - 5 Marks

[ ]:
```

```

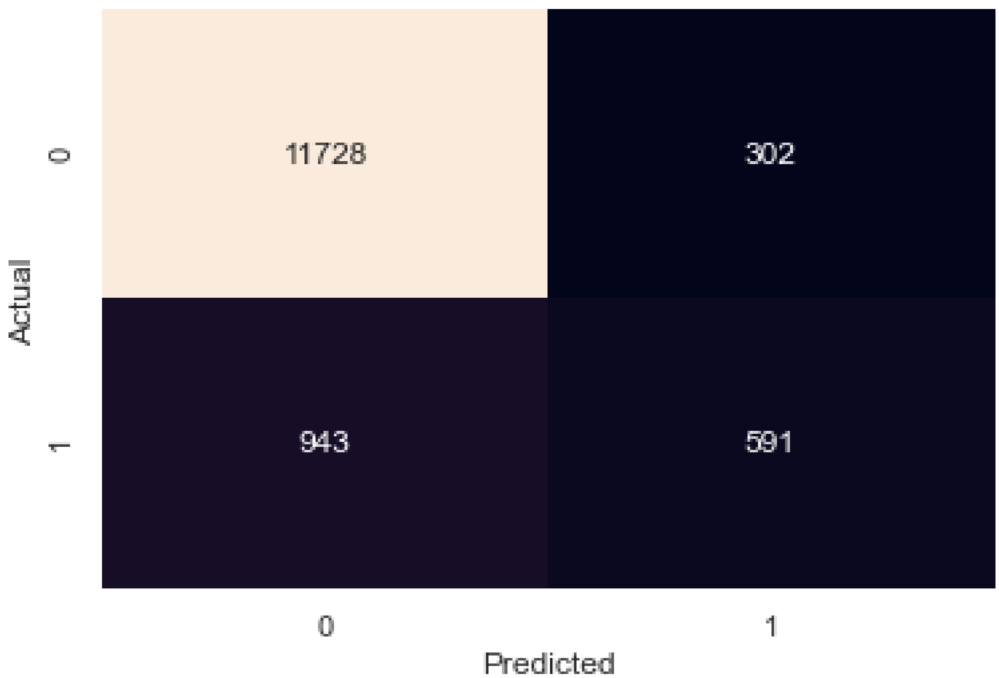
5.3 Random Forest

[25]:
```
model = RandomForestClassifier(n_estimators = 50)
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
```

Also reflect the training and testing score of all the models. Build a dataframe with model names as row index and all the metrics calculated as columns - 5 Marks

[26]:
```
printReports(y_test, x_test, y_train, x_train, model)
```

```
              precision    recall  f1-score   support

           0       0.93      0.97      0.95     12030
           1       0.66      0.39      0.49      1534

    accuracy                           0.91     13564
   macro avg       0.79      0.68      0.72     13564
weighted avg       0.90      0.91      0.90     13564
```




```
***TRAINING DATA***
precision score:  1.0
recall score:  0.9986684420772304
accuracy score:  0.9998420071412772
ROC AUC score:  0.9993342210386151
F1 score:  0.999333777481679


***TESTING DATA***
precision score:  0.6618141097424413
recall score:  0.3852672750977836
accuracy score:  0.9082129165437924
ROC AUC score:  0.6800816840991827
F1 score:  0.4870210135970333
```

```
***CROSS VALIDATION DATA***
precision score  0.63
recall score  0.36
accuracy score  0.91
roc_auc score  0.92
f1 score  0.44
```

[27]:
```
featureImportance(model)
```

	feature_rating	features
0	0.267074	duration
1	0.103010	balance
2	0.099096	age
3	0.088786	day
4	0.057034	poutcome_success
5	0.040905	campaign
6	0.040727	pdays
7	0.022642	housing_yes
8	0.021851	previous
9	0.015895	contact_telephone
10	0.013924	month_mar
11	0.013393	education_secondary
12	0.012336	marital_married
13	0.011792	education_tertiary
14	0.011581	month_jun
15	0.011500	month_oct
16	0.011113	month_may
17	0.011042	job_technician
18	0.010613	month_aug
19	0.010602	job_management
20	0.010280	marital_single
21	0.010188	month_jul
22	0.010155	loan_yes
23	0.009435	month_sep
24	0.009287	job_blue-collar
25	0.008753	month_nov
26	0.008685	month_feb
27	0.008453	poutcome_unknown
28	0.007154	job_services
29	0.006224	month_jan
30	0.005603	job_retired
31	0.004830	job_unemployed
32	0.004757	job_student
33	0.004723	job_self-employed
34	0.004008	month_dec
35	0.003895	job_entrepreneur

```
36         0.003682       poutcome_other
37         0.003281        job_housemaid
38         0.001692          default_yes
```

[ ]:

```

```

5.4 Boosting

5.4.1 adaboost

[28]:

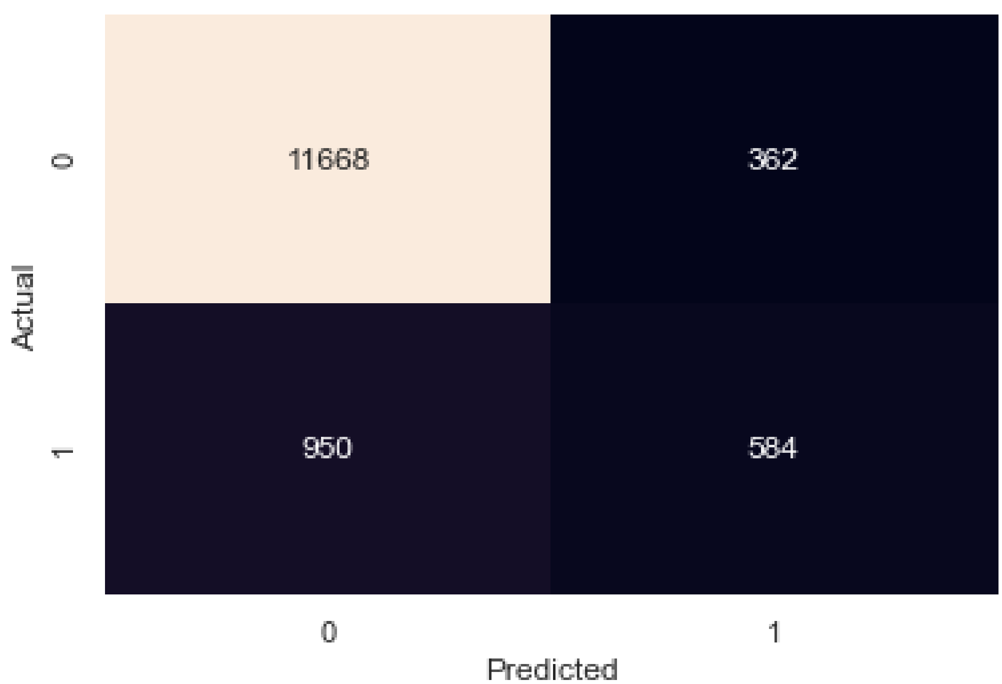
```
model = AdaBoostClassifier()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
```

Also reflect the training and testing score of all the models. Build a dataframe with model names as row index and all the metrics calculated as columns - 5 Marks

[29]:

```
printReports(y_test, x_test, y_train, x_train, model)
```

```
              precision    recall  f1-score   support

           0       0.92      0.97      0.95     12030
           1       0.62      0.38      0.47      1534

    accuracy                           0.90     13564
   macro avg       0.77      0.68      0.71     13564
weighted avg       0.89      0.90      0.89     13564
```



```
***TRAINING DATA***
precision score:  0.6242263483642794
recall score:  0.3760319573901465
accuracy score:  0.8991057604196291
ROC AUC score:  0.6727786346537711
F1 score:  0.4693368788432774


***TESTING DATA***
precision score:  0.6173361522198731
recall score:  0.38070404172099087
accuracy score:  0.903273370687113
ROC AUC score:  0.6753063018247514
F1 score:  0.47096774193548385


***CROSS VALIDATION DATA***
precision score  0.62
recall score  0.38
accuracy score  0.90
roc_auc score  0.90
f1 score  0.47
```

[30]:

```
featureImportance(model)
```

	feature_rating	features
0	0.30	duration
1	0.10	day
2	0.08	age
3	0.06	housing_yes
4	0.04	campaign
5	0.04	pdays
6	0.04	poutcome_success
7	0.02	month_may
8	0.02	education_tertiary
9	0.02	month_feb
10	0.02	month_jan
11	0.02	month_jul
12	0.02	balance
13	0.02	month_jun
14	0.02	month_mar
15	0.02	job_blue-collar
16	0.02	month_oct
17	0.02	month_sep
18	0.02	month_dec
19	0.02	poutcome_other

```
20            0.02      marital_married
21            0.02             loan_yes
22            0.02             previous
23            0.02    contact_telephone
24            0.00            month_nov
25            0.00          job_student
26            0.00            month_aug
27            0.00  education_secondary
28            0.00       job_unemployed
29            0.00       job_technician
30            0.00         job_services
31            0.00    job_self-employed
32            0.00          job_retired
33            0.00       job_management
34            0.00        job_housemaid
35            0.00     job_entrepreneur
36            0.00       marital_single
37            0.00          default_yes
38            0.00     poutcome_unknown
```

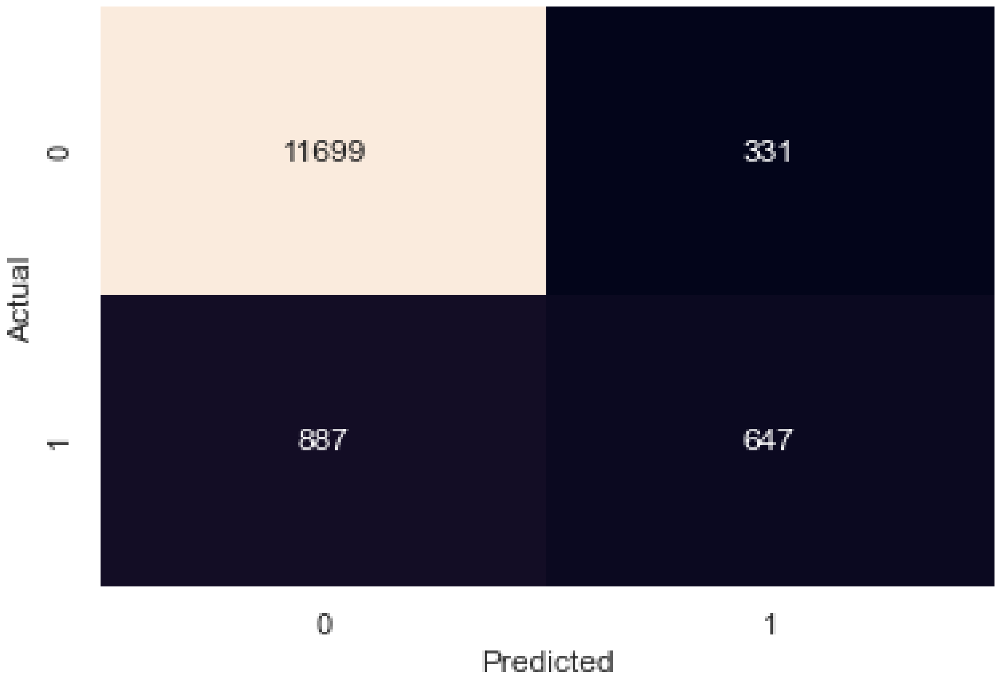
[ ]:

5.4.2 gradient descent

[31]:
```
model = GradientBoostingClassifier()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
```

[32]:
```
printReports(y_test, x_test, y_train, x_train, model)
```

```
              precision    recall  f1-score   support

           0       0.93      0.97      0.95     12030
           1       0.66      0.42      0.52      1534

    accuracy                           0.91     13564
   macro avg       0.80      0.70      0.73     13564
weighted avg       0.90      0.91      0.90     13564
```




```
***TRAINING DATA***
precision score:  0.6937285223367697
recall score:  0.43009320905459386
accuracy score:  0.9098492748127784
ROC AUC score:  0.7022651618197105
F1 score:  0.5309879993424297


***TESTING DATA***
precision score:  0.6615541922290389
recall score:  0.4217731421121252
accuracy score:  0.9102034797994691
ROC AUC score:  0.6971292975731033
F1 score:  0.5151273885350318


***CROSS VALIDATION DATA***
precision score  0.64
recall score  0.41
accuracy score  0.91
roc_auc score  0.92
f1 score  0.50
```

[33]:
```
featureImportance(model)
```

	feature_rating	features
0	0.471013	duration
1	0.202357	poutcome_success
2	0.045210	housing_yes
3	0.041249	age
4	0.039165	pdays
5	0.036536	month_mar
6	0.029593	contact_telephone
7	0.023230	month_oct
8	0.022587	day
9	0.021765	month_jun
10	0.015775	month_sep
11	0.010063	balance
12	0.005949	campaign
13	0.004034	marital_married
14	0.003942	month_nov
15	0.003669	month_may
16	0.003478	month_dec
17	0.003096	education_tertiary
18	0.002627	job_student
19	0.002538	loan_yes
20	0.002005	month_feb
21	0.001918	month_jan
22	0.001529	month_jul
23	0.001214	previous
24	0.001135	month_aug
25	0.001045	job_entrepreneur
26	0.000882	job_blue-collar
27	0.000761	poutcome_unknown
28	0.000633	job_housemaid
29	0.000290	poutcome_other
30	0.000265	marital_single
31	0.000227	job_technician
32	0.000219	job_management
33	0.000000	job_retired
34	0.000000	job_self-employed
35	0.000000	default_yes
36	0.000000	education_secondary
37	0.000000	job_unemployed
38	0.000000	job_services

5.4.3 Also reflect the training and testing score of all the models. Build a dataframe with model names as row index and all the metrics calculated as columns - 5 Marks

[34]:
```
results = pd.DataFrame(columns = columns, index = models)
results['Testing Precision'] = precision_test
results['Testing Recall'] = recall_test
results['Testing Accuracy'] = accuracy_test
results['Testing ROC_AUC'] = roc_auc_test
results['Testing F1'] = f1_test

results['Training Precision'] = precision_train
results['Training Recall'] = recall_train
results['Training Accuracy'] = accuracy_train
results['Training ROC_AUC'] = roc_auc_train
results['Training F1'] = f1_train

results['CrossValidation Precision'] = precision_xval
results['CrossValidation Recall'] = recall_xval
results['CrossValidation Accuracy'] = accuracy_xval
results['CrossValidation ROC_AUC'] = roc_auc_xval
results['CrossValidation F1'] = f1_xval

round(results.head(),5)
```

[34]:
```
               Testing Precision  Testing Recall  Testing Accuracy  \
Decision Tree            0.48383         0.48761           0.88322
Bagging                  0.59696         0.40939           0.90195
Random Forest            0.66181         0.38527           0.90821
Adaboost                 0.61734         0.38070           0.90327
Gradientboost            0.66155         0.42177           0.91020

               Testing ROC_AUC  Testing F1  Training Precision  \
Decision Tree          0.71064     0.48571             0.92007
Bagging                0.68707     0.48569             0.99406
Random Forest          0.68008     0.48702             1.00000
Adaboost               0.67531     0.47097             0.62423
Gradientboost          0.69713     0.51513             0.69373

               Training Recall  Training Accuracy  Training ROC_AUC  \
Decision Tree          0.90120            0.97899           0.94533
Bagging                0.93662            0.99182           0.96793
Random Forest          0.99867            0.99984           0.99933
Adaboost               0.37603            0.89911           0.67278
Gradientboost          0.43009            0.90985           0.70227

               Training F1  CrossValidation Precision  CrossValidation Recall  \
Decision Tree      0.91053                    0.46295                 0.44592
```

Bagging	0.96449	0.58147	0.37611
Random Forest	0.99933	0.62950	0.35785
Adaboost	0.46934	0.62039	0.38001
Gradientboost	0.53099	0.63972	0.40800

	CrossValidation Accuracy	CrossValidation ROC_AUC	\
Decision Tree	0.87828	0.70910	
Bagging	0.90173	0.87818	
Random Forest	0.90674	0.91587	
Adaboost	0.90342	0.90443	
Gradientboost	0.90696	0.91830	

	CrossValidation F1
Decision Tree	0.44707
Bagging	0.46320
Random Forest	0.44443
Adaboost	0.46993
Gradientboost	0.49716

5.4.4 Explain the confusion matrix related terms like recall, precision etc. Also, select the best metric to choose one of the models from above. Give your reason for the same. - 5 Marks

Recall is the percentage of capturable market. One can think of recall as missed oportunities. tp/(tp+fn)

Precision grades the certanty of the model. one can think of precision as risk or uncertainty. tp/(tp+fp)

accuracy score is (tp + tn) /(tp + tn + fp + fn)

5.4.5 Answer the following questions : - 10 Marks

What do you mean by recall and what information does it provide here? recall here provides us the total amount of people who would purchase our product

Suggest some changes for the organization so that they can increase the number of customers who take term deposit. most models identified previous contact duration, previous campaign success and balance were shown in multiple models to have strongest correlation

How much influence does the previous campaign and mode of interaction have on financial performance. previous campaign results played a large part in predicting future purchases. Mode of interaction played a much smaller role.

Which features should be more/less focused by the bank to get better results and why? most models identified previous contact duration, previous campaign success and balance were shown in multiple models to have strongest correlation

What did you learn about banking industries from this data? Banks like to sell products

```
[ ]:
```

Note : Use random_state=7 (wherever the parameter can be used) so that we can compare all submissions.

Provide comments in the solution notebook regarding the steps you take and also provide insights drawn from the plots. - 5 Marks.

Marks distribution for Students with recall_score (pos_label = 'yes') on the test set:

Above 43% - 5 Marks

Between 40% to 43% - 4 Marks

Less than 40% - 3 Marks

```
[ ]:
```