Allison Roeser

Bank Marketing Study: Evaluating Classification Models

<u>Data Preparation, Exploration, Visualization</u>:

The objective of the Bank Marketing study was to determine the set of customers most likely to respond to a marketing campaign for a new term deposit. The dataset used included 17 variables for 4,521 customers. The variables include demographic information (such as age, education level, and marital status), as well as, information about activity as a bank customer (such as if the customer has a personal loan). The response to the new term deposit campaign for each unique customer observation is the target variable being modeled.

Figure 1 shows boxplots for the six continuous, potential predictor variables after they have each been scaled using Standard Scaling. The distribution differences between responses are especially evident for the variables *duration*, *pdays*, and *previous*. Figure 2 includes bar charts for the categorical variables grouped by response type. Response differences between categories are most evident in the categorical variable "poutcome" (the response to a prior marketing campaign). If the prior marketing campaign was a success, the customer is more likely to respond positively than negatively to the new term deposit campaign.

Correlations for the three binary predictor variables and *response* are shown in Figure 4. *Housing* has the highest correlation with *response* at -0.105. *Loan*, *housing*, and *default* are weak predictor variables. Confusion matrix bar charts for the three binary variables are shown in Figure 5. Customers without a housing loan are more likely to respond positively to the campaign.

All potential predictor variables having a correlation to *response* greater than 0.10 are displayed in Figure 5. The continuous variables *duration* and *poutcome_success* are the most highly correlated with *response*. *Duration*, which has a correlation of 0.401 with *response*, measures the length of time in seconds of the last contact with the customer. It appears that customers willing to stay on the phone previously are more likely to respond well to the new campaign.

<u>Implementation and Programming</u>:

Full code is attached in the Appendix to show the specific implementation. Scikit-learn was leveraged for regression model building using Logistic Regression and Naive Bayes Regression techniques. Pandas and Seaborn were leveraged for exploratory data analysis and visualization. Predictor variables were chosen based on correlation to the response variables as evidenced through correlation matrices and visual plotting of variables. Models were evaluated using ROC curves and AUC scores.

Review Research Design and Modeling Methods:

Logistic Regression and Naive Bayes Regression models were run to build models to accurately predict the probability that the categorical dependent variable *response* is 1 (representing a response of yes). When evaluating potential predictor variables, correlation with *response* was the main criteria used. Predictor variables were culled to a manageable size in order to make the regression equations more interpretable and the insights more actionable for management. All continuous, predictor variables were scaled using Standard Scaling to ensure proper weighting of variable importance. Categorical variables with values of yes/no were transformed to having values of 1/0. For categorical variables with more than two categories, new binary variables were created for each category value (*poutcome_success* from *poutcome*).

AUC scores and ROC curves were used to evaluate models. ROC curves are useful when evaluating classification of binary variables because ROC curves provide a simple way to compare the True Positive Rate of a model with the False Positive Rate. The True Positive Rate (also known as Recall or Sensitivity) is how well the model correctly identifies positive *responses.* In contrast, the False Positive Rate is when the model incorrectly identifies a negative *response* as a positive *response.* AUC scores range from 0 to 1. An AUC of 0.5 is the baseline for a random model. The closer the score is to 1, the better the model is at correctly classifying *responses*.

Review Results, Evaluate Models:

The final models were run using six predictor variables: duration, poutcome_success, previous, housing, loan, and default. Both models preformed similarly on the selected variables with the Logistic model having an AUC score of 0.771 the Naive Bayes model scoring 0.754. ROC curves for each model are displayed in Figures 6 & 7.

The Naive Bayes model achieved a True Positive Rate (Sensitivity) of 77.6% versus 70.9% for the Logistic Regression model (Figures 8 & 9). The cost of a higher TPR for the Naive Bayes model was also a higher FPR. The FPR rate for the Naive Bayes model was 26.8% versus 16.6% for Logistic Regression model. Naive Bayes correctly identified an additional 28 *responses* as positive; however, the tradeoff was that the Bayes model also misclassified an additional 325 *responses* as positive when they were actually negative (Figures 10 & 11). The Logistic Regression model had a higher TNR (Specificity) at 83.4% versus 73.2% for the Logistic model. The tradeoff between Sensitivity and Specificity is seen when choosing between the models. Since the cost of misclassification is relatively low when modeling responses to a marketing campaign, both models perform well and provide valuable insights.

<u>Exposition, Problem Description and Management Recommendations</u>:

The continuous variable *duration* and the categorical variable *poutcome_success* both proved to be important predictors of response probability. The bank should target customers who previously had longer than average interactions with bank sales staff. The bank should also target customers for whom the previous marketing campaign was successful. Customers who did not have a home loan with the bank were more likely to respond positively to the marketing for the term loan. Therefore, the bank should target customers not currently holding a housing loan. Because housing was a weak predictor, the bank should only use this predictor in conjunction with other positive indicators of success.

The Logistic Regression model preformed slightly better than the Native Bayes model using the AUC scoring criteria. The Logistic Regression model had a higher Specificity but lower Sensitivity. The bank should determine whether they want to maximize True Negatives or True Positives. The Logistic models higher Specificity allows it to better identify individuals who would not be interested in the new term loan (True Negatives). This would be used if the bank wanted to make sure they did not waste time and money marketing to individuals who would not be interested. In contrast, Naive Bayes' higher Sensitivity identifies more individuals who would be interested in the loan, but the higher Sensitivity comes with the cost of much more False Positives.

References

Holtz, Y. #34 Grouped boxplot. (n.d.). The Python Graph Gallery. Retrieved from: https://python-graph-gallery.com/34-grouped-boxplot/

Kunanbaeva, A. What is a ROC AUC and how to visualize it in python. (Sep 4, 2019). Medium. Retrieved from: https://medium.com/@kunanba/what-is-roc-auc-and-how-to-visualize-it-in-python-f35708206663

Li, S. Building a logistic regression in python, step-by-step. (Sep 28, 2017). Towards Data Science. Retrieved from: https://towardsdatascience.com/building-a-logistic-regression-in-python-step-by-step-becd4d56c9c8

Appendix

Figure 1: Boxplots for continuous, predictor variables segmented by response (Holtz)

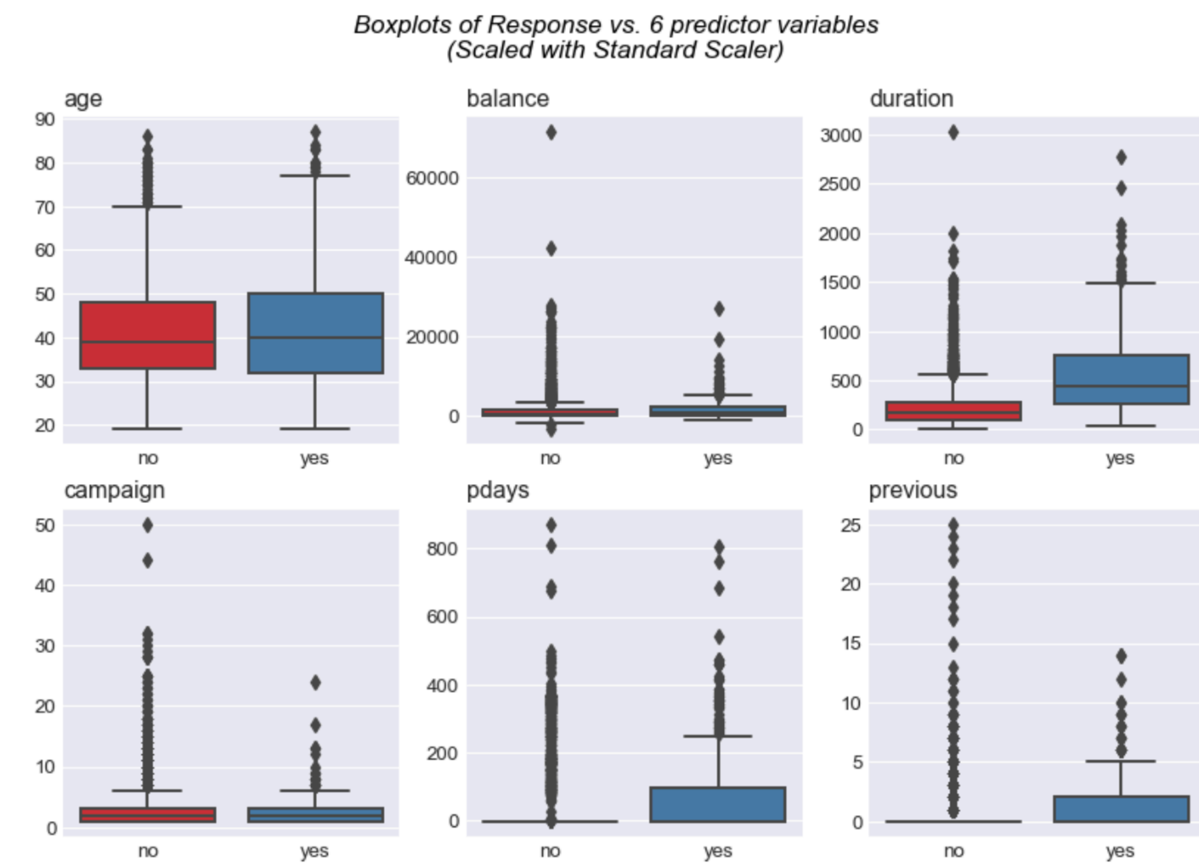


Figure 2: Bar charts for categorical predictor variables segmented by response

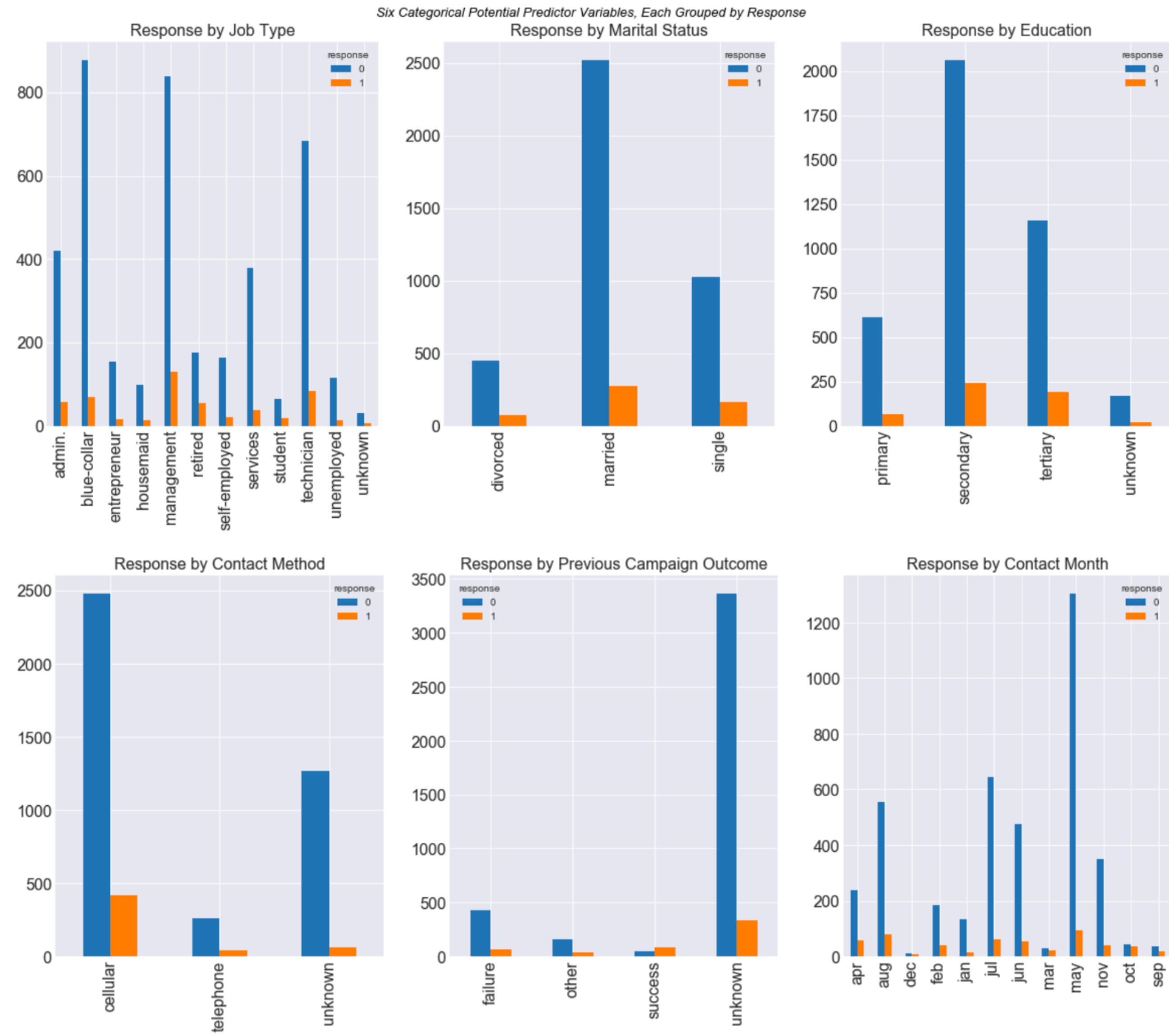


Figure 3: Correlation matrix of response and binary variables

	response	default	housing	loan
response	1.000000	0.001303	-0.104683	-0.070517
default	0.001303	1.000000	0.006881	0.063994
housing	-0.104683	0.006881	1.000000	0.018451
loan	-0.070517	0.063994	0.018451	1.000000

Figure 4: Housing, default and loan binary variables segmented by response

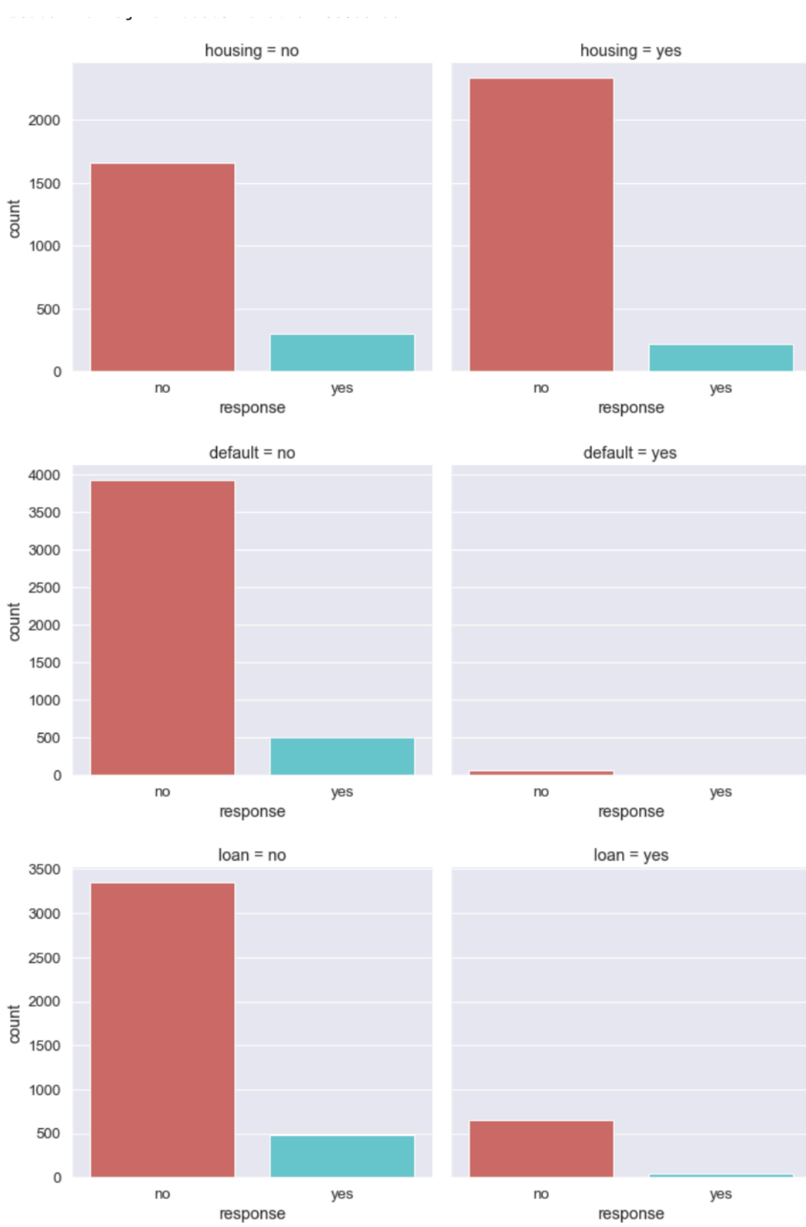


Figure 5: Potential predictor variables' correlation with response

```
housing             0.104683
duration            0.401118
pdays               0.104087
previous            0.116714
response            1.000000
contact_cellular    0.118761
contact_unknown     0.139399
month_mar           0.102716
month_may           0.102077
month_oct           0.145964
poutcome_success    0.283481
poutcome_unknown    0.162038
Name: response, dtype: float64
```

Figure 6: ROC curve for Logistic Regression model (Li 2017)

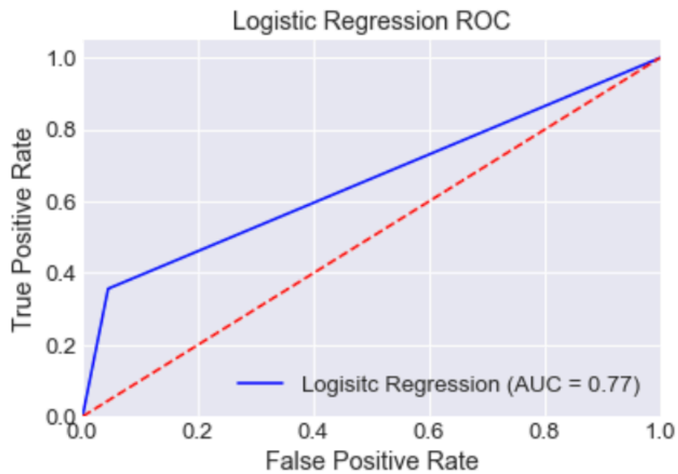


Figure 7: ROC curve for Naive Bayes Regression model (Li 2017)

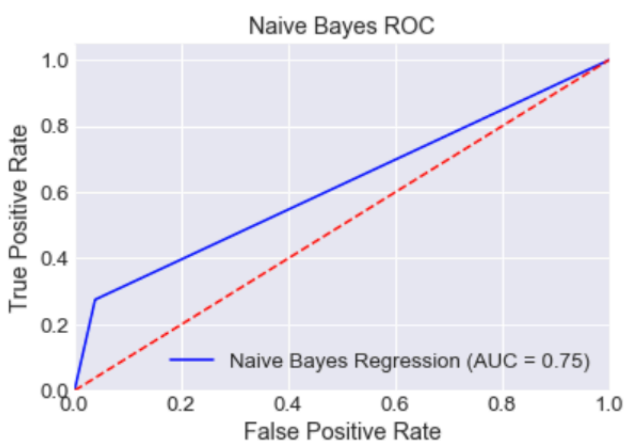


Figure 8: Logistic Regression Summary Statistics

```
Logistic Regression
------------------------
True Positives: 297
False Positives: 532
True Negatives: 2670
False Negatives: 122
-------------------------
True Positive Rate (Sensitivity): 0.709
False Positives: 0.166
True Negatives (Specificity): 0.834
False Negatives: 0.291
------------------------
Area Under the Curve: 0.771
```

Figure 9: Naive Bayes Summary Statistics

```
Naive Bayes
------------------------
True Positives: 325
False Positives: 857
True Negatives: 2345
False Negatives: 94
------------------------
True Positive Rate (Sensitivity): 0.776
False Positives: 0.268
True Negatives (Specificity): 0.732
False Negatives: 0.224
------------------------
Area Under the Curve: 0.754
```

Figure 10: Logistic Regression confusion matrix

	Predicted Response No	Predicted Response Yes
Actual Response No	2670	532
Actual Response Yes	122	297

Figure 11: Naive Bayes confusion matrix

	Predicted Response No	Predicted Response Yes
Actual Response No	2670	532
Actual Response Yes	122	297

In [1]:

```
# Jump-Start for the Bank Marketing Study
# as described in Marketing Data Science: Modeling Techniques
# for Predictive Analytics with R and Python (Miller 2015)

# jump-start code revised by Thomas W. Milller (2018/10/07)

# Scikit Learn documentation for this assignment:
# http://scikit-learn.org/stable/auto_examples/classification/
#   plot_classifier_comparison.html
# http://scikit-learn.org/stable/modules/generated/
#   sklearn.naive_bayes.BernoulliNB.html#sklearn.naive_bayes.BernoulliNB.sc
# http://scikit-learn.org/stable/modules/generated/
#   sklearn.linear_model.LogisticRegression.html
# http://scikit-learn.org/stable/modules/model_evaluation.html
# http://scikit-learn.org/stable/modules/generated/
#  sklearn.model_selection.KFold.html

# prepare for Python version 3x features and functions
# comment out for Python 3.x execution
# from __future__ import division, print_function
# from future_builtins import ascii, filter, hex, map, oct, zip

# seed value for random number generators to obtain reproducible results
RANDOM_SEED = 1

# import base packages into the namespace for this program
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score as cvs
from sklearn.linear_model import LogisticRegression as lr
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.naive_bayes import BernoulliNB as bern
import seaborn as sns
import scipy.stats as stats
import random
import sklearn.utils.validation as val
from sklearn.utils import resample
from sklearn.metrics import roc_curve, auc
import statistics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score

# initial work with the smaller data set
bank = pd.read_csv('bank.csv', sep = ';')  # start with smaller data set
# examine the shape of original input data
print(bank.shape)
```

```
(4521, 17)
```

In [2]:
```
# drop observations with missing data, if any
bank.dropna()
# examine the shape of input data after dropping missing data
print(bank.shape)
```

```
(4521, 17)
```

In [3]:
```
# look at the list of column names, note that y is the response
list(bank.columns.values)
```

Out[3]:
```
['age',
 'job',
 'marital',
 'education',
 'default',
 'balance',
 'housing',
 'loan',
 'contact',
 'day',
 'month',
 'duration',
 'campaign',
 'pdays',
 'previous',
 'poutcome',
 'response']
```

In [4]:
```
bank.info()
```

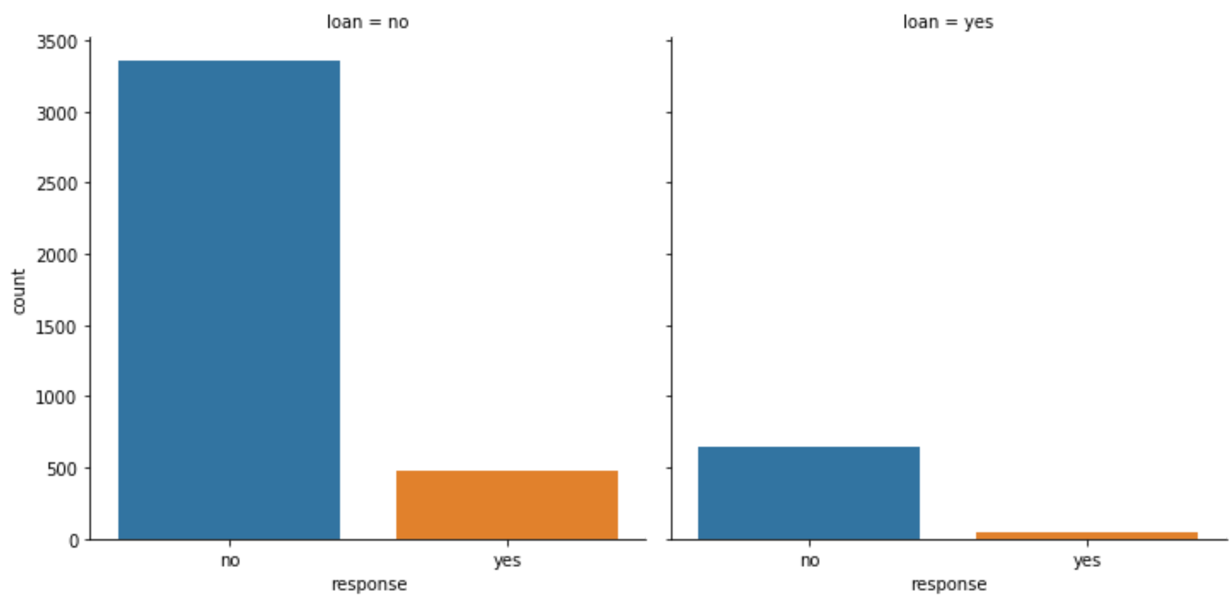
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4521 entries, 0 to 4520
Data columns (total 17 columns):
age          4521 non-null int64
job          4521 non-null object
marital      4521 non-null object
education    4521 non-null object
default      4521 non-null object
balance      4521 non-null int64
housing      4521 non-null object
loan         4521 non-null object
contact      4521 non-null object
day          4521 non-null int64
month        4521 non-null object
duration     4521 non-null int64
campaign     4521 non-null int64
pdays        4521 non-null int64
previous     4521 non-null int64
poutcome     4521 non-null object
response     4521 non-null object
dtypes: int64(7), object(10)
memory usage: 600.6+ KB
```

In [5]:
```
# look at the beginning of the DataFrame
bank.head()
```

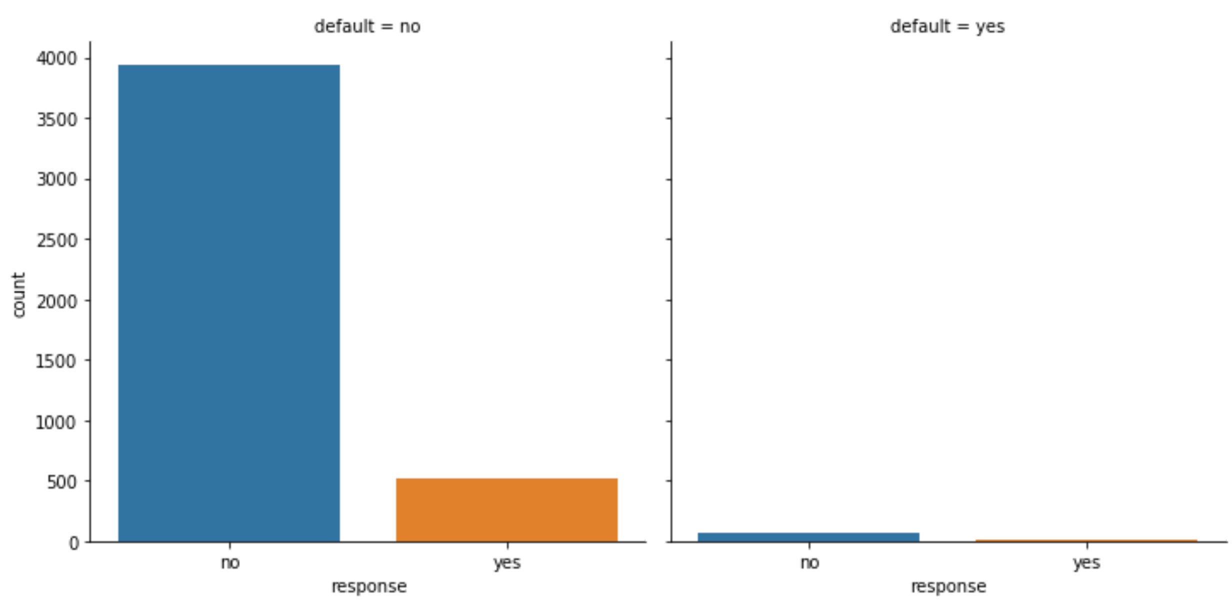
Out[5]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	du
0	30	unemployed	married	primary	no	1787	no	no	cellular	19	oct	
1	33	services	married	secondary	no	4789	yes	yes	cellular	11	may	
2	35	management	single	tertiary	no	1350	yes	no	cellular	16	apr	
3	30	management	married	tertiary	no	1476	yes	yes	unknown	3	jun	
4	59	blue-collar	married	secondary	no	0	yes	no	unknown	5	may	

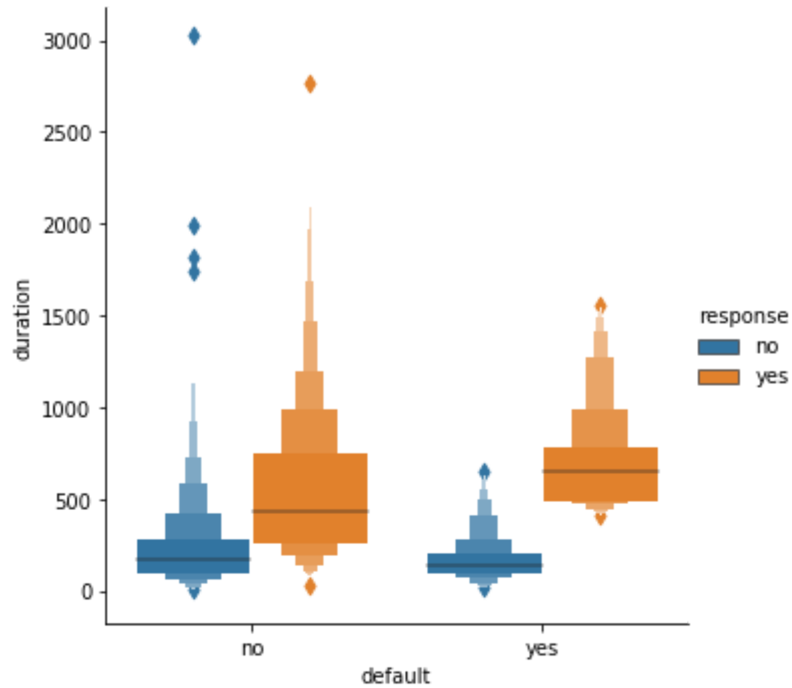
In [6]:
```
fig1 = sns.catplot(x = "response", col = "loan", data = bank, kind = "count
```



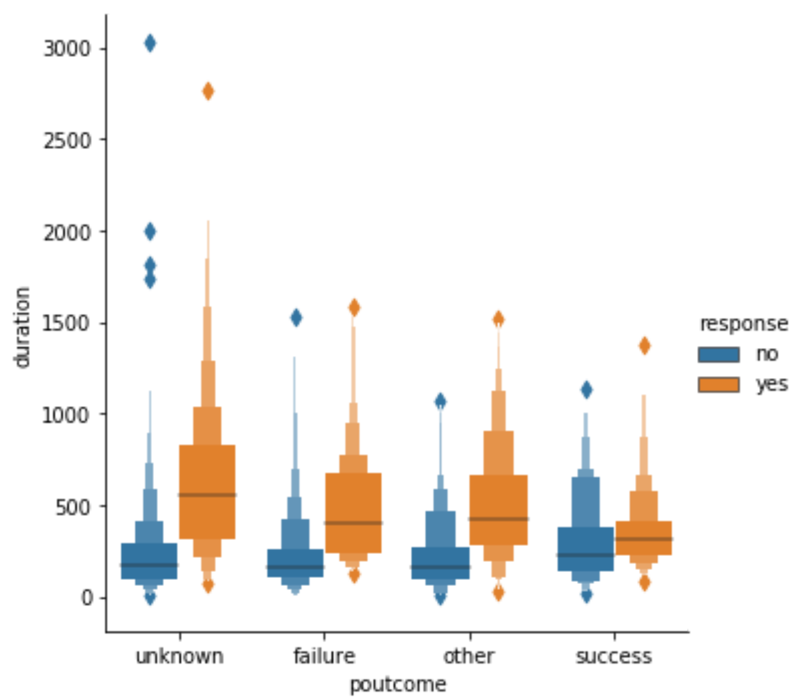
In [7]:
```
fig1 = sns.catplot(x = "response", col = "default", data = bank, kind = "cc
```



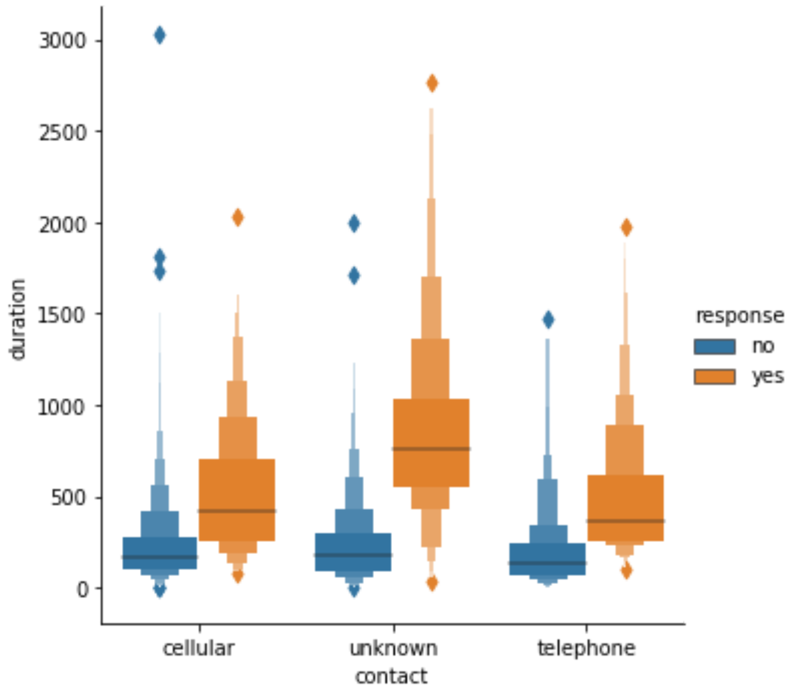
In [8]:
```
fig1 = sns.catplot(x="default", y = "duration", hue = "response", kind = "b
```



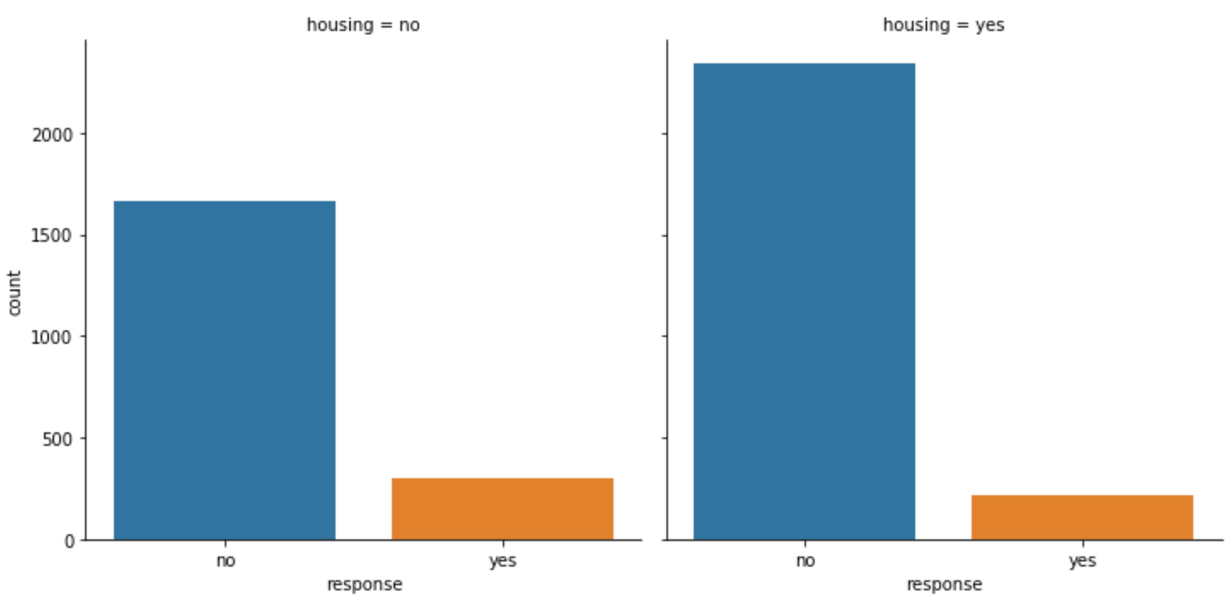
In [9]:
```
fig1 = sns.catplot(x="poutcome", y = "duration", hue = "response", kind = "
```



In [10]:
```
fig1 = sns.catplot(x="contact", y = "duration", hue = "response", kind = "b
```

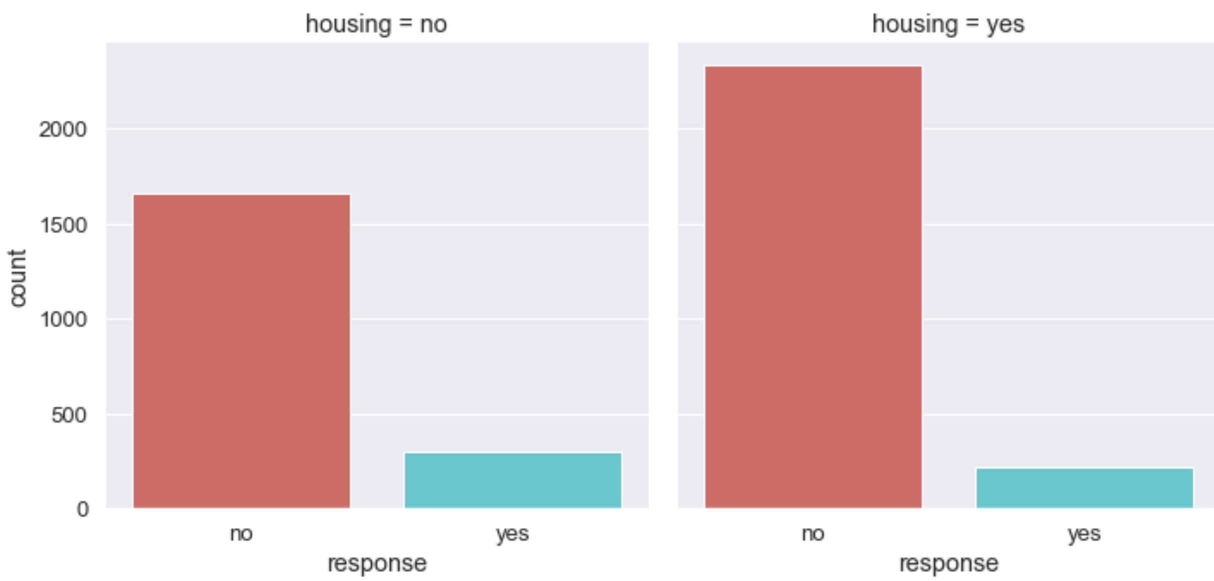


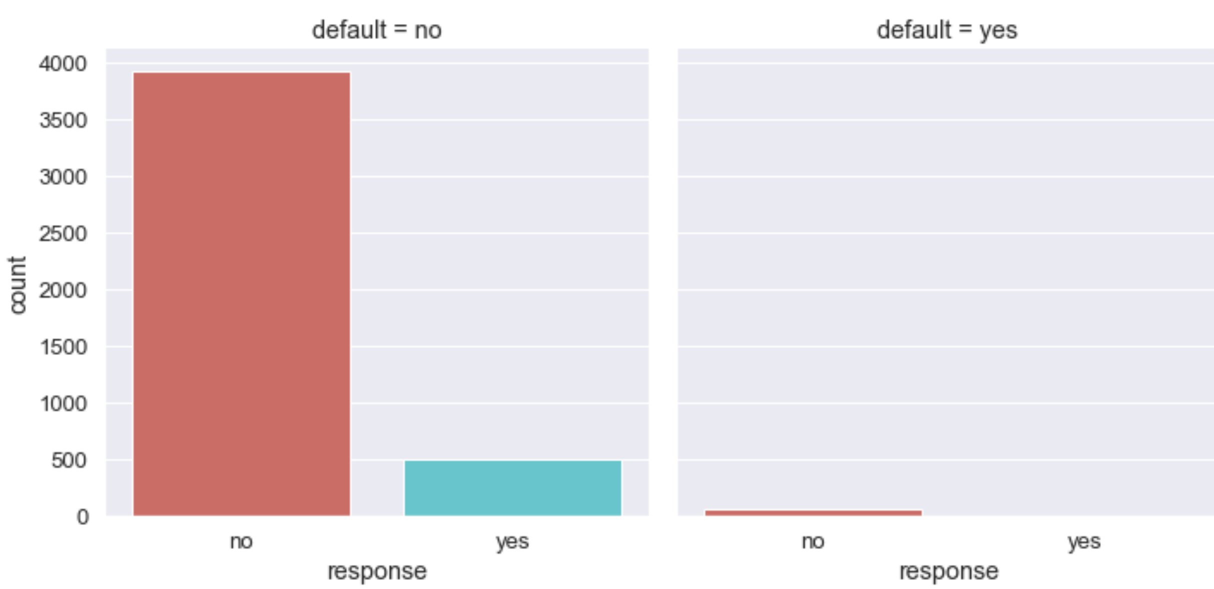
In [11]:
```
fig1 = sns.catplot(x = "response", col = "housing", data = bank, kind = "cc
```

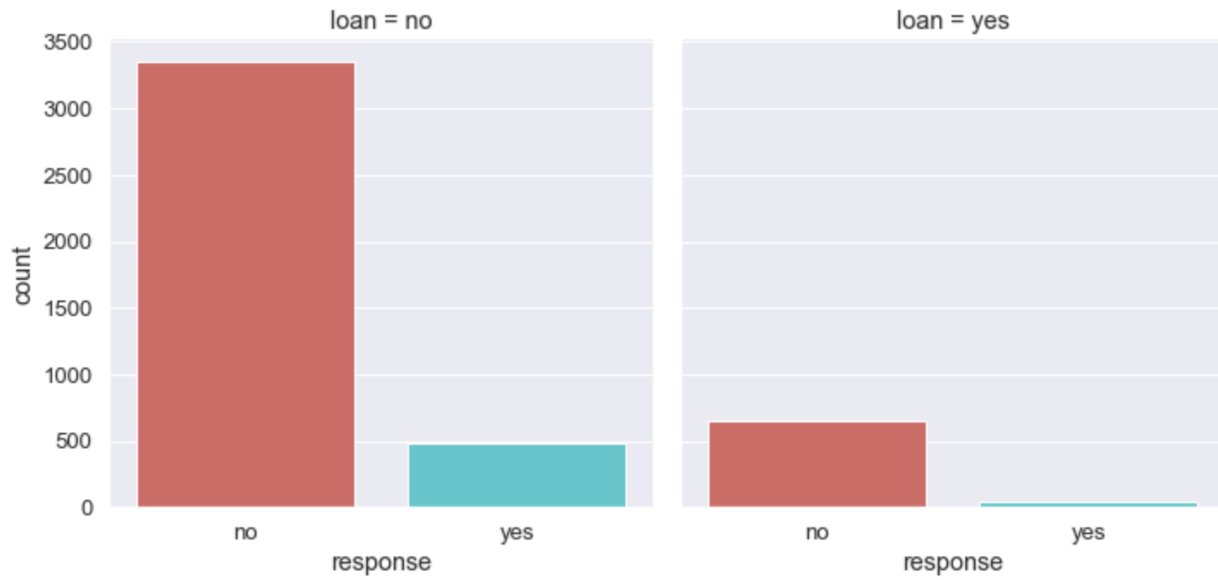


In [12]:
```
sns.set(font_scale=1.2)
sns.catplot(x = "response", col = "housing", data = bank, kind = "count", p
sns.catplot(x = "response", col = "default", data = bank, kind = "count", p
sns.catplot(x = "response", col = "loan", data = bank, kind = "count", pale
```

Out[12]: <seaborn.axisgrid.FacetGrid at 0x12c4d44d0>



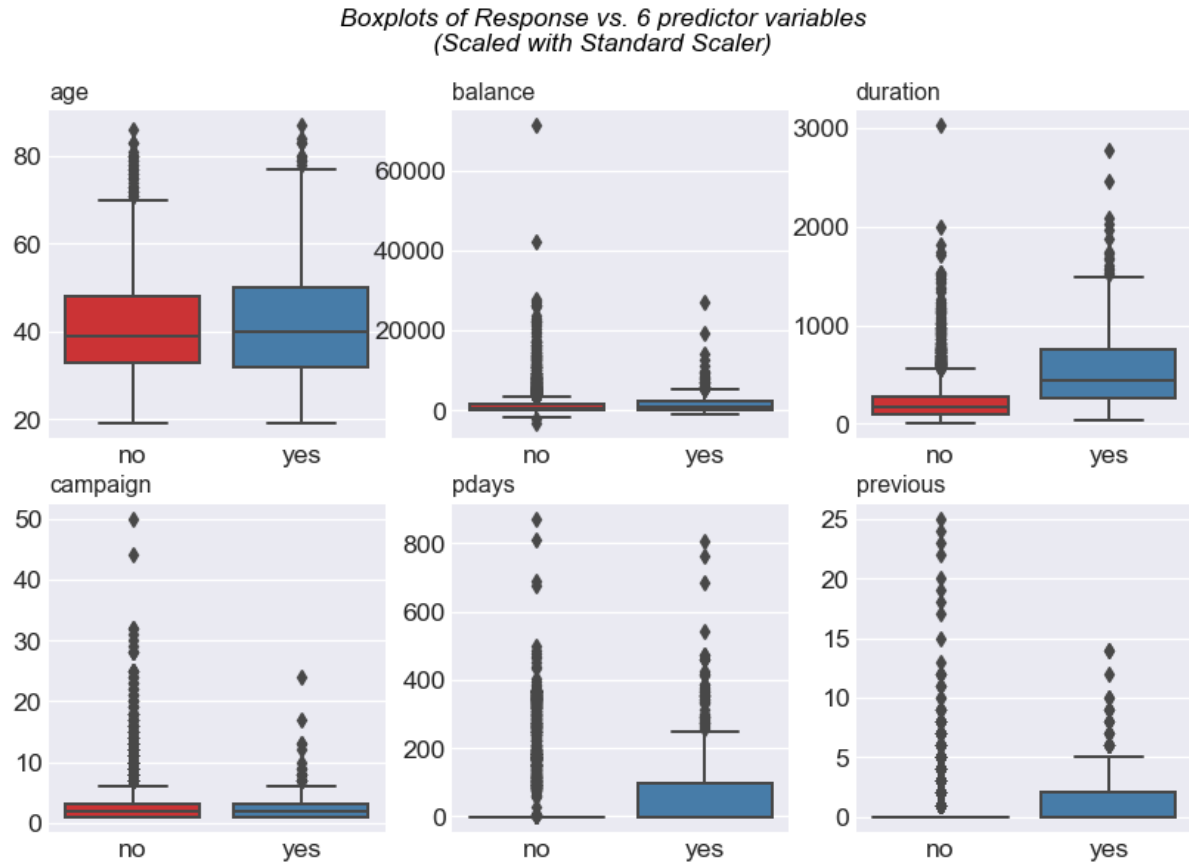




In [13]:
```
bank_scatter = pd.DataFrame(data = bank, columns = ["age", "balance", "dura
```

In [14]:
```
plt.style.use('seaborn-darkgrid')
my_dpi=96
plt.figure(figsize=(1000/my_dpi, 1000/my_dpi), dpi=my_dpi)

# create a color palette
#palette = plt.get_cmap('Set1')

# multiple line plot
num=0
for column in bank_scatter.drop('response', axis=1):
    num+=1

    # Find the right spot on the plot
    plt.subplot(3,3, num)

    # Plot the lineplot
    #plt.plot( y=relevant["mv", relevant[column]])
    sns.boxplot(x="response", y =column,  data=bank_scatter, palette="Set1"
    #plt.plot(relevant['mv'], relevant[column], marker='', color=palette(nu


    # Not ticks everywhere
    if num in range(10) :
        #plt.tick_params(labelbottom='off')
        plt.ylabel('')
        plt.xlabel('')
    if num not in [1,4,7] :
        plt.tick_params(labelleft='off')

    # Add title
    plt.title(column, loc='left', fontsize=12, fontweight=0 )

# general title
plt.suptitle("Boxplots of Response vs. 6 predictor variables\n(Scaled with 

# Axis title
#plt.text(-4, -3, 'Standardized Scale', ha='center', va='center')
#plt.text(-13, 7, 'Median Home Value (Standardized Scale)', ha='center', va
#plt.text(-6.2, 10.1, 'tax', ha='center', va='center')
```

Out[14]:
```
Text(0.5, 0.95, 'Boxplots of Response vs. 6 predictor variables\n(Scaled
with Standard Scaler)')
```



In [15]:
```
# mapping function to convert text no/yes to integer 0/1
convert_to_binary = {'no' : 0, 'yes' : 1}
```

In [16]:
```
# define binary variable for having credit in default
bank["default"] = bank['default'].map(convert_to_binary)
```

In [17]:
```
# define binary variable for having a mortgage or housing loan
bank["housing"] = bank['housing'].map(convert_to_binary)
```

In [18]:
```
# define binary variable for having a personal loan
bank["loan"] = bank['loan'].map(convert_to_binary)
```

In [19]:
```
# define response variable to use in the model
bank["response"] = bank['response'].map(convert_to_binary)
```

In [20]:
```
# gather three explanatory variables and response into a numpy array
# here we use .T to obtain the transpose for the structure we want
#model_data = np.array([np.array(default), np.array(housing), np.array(loan
    #np.array(response)]).T
```

In [21]:
```
# examine the shape of model_data, which we will use in subsequent modeling
#print(model_data.shape)
```

In [22]:
```
# the rest of the program should set up the modeling methods
# and evaluation within a cross-validation design
```

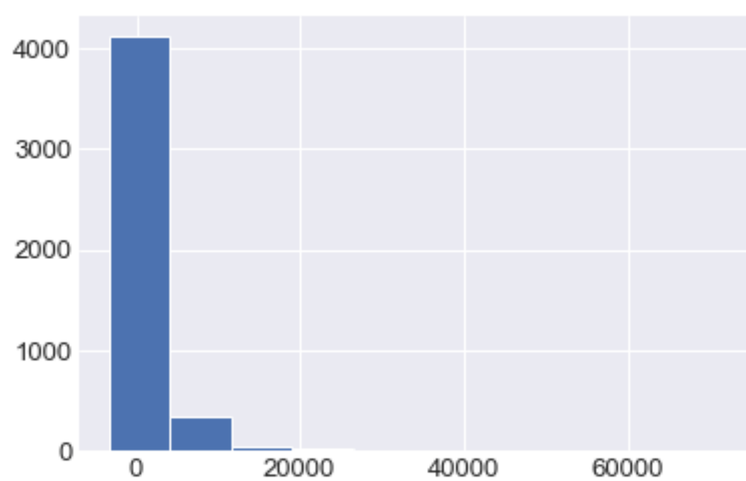
In [23]:
```
bank.describe()
```

Out[23]:

	age	default	balance	housing	loan	day	duration
count	4521.000000	4521.000000	4521.000000	4521.000000	4521.000000	4521.000000	4521.000000
mean	41.170095	0.016810	1422.657819	0.566025	0.152842	15.915284	263.961292
std	10.576211	0.128575	3009.638142	0.495676	0.359875	8.247667	259.856633
min	19.000000	0.000000	-3313.000000	0.000000	0.000000	1.000000	4.000000
25%	33.000000	0.000000	69.000000	0.000000	0.000000	9.000000	104.000000
50%	39.000000	0.000000	444.000000	1.000000	0.000000	16.000000	185.000000
75%	49.000000	0.000000	1480.000000	1.000000	0.000000	21.000000	329.000000
max	87.000000	1.000000	71188.000000	1.000000	1.000000	31.000000	3025.000000

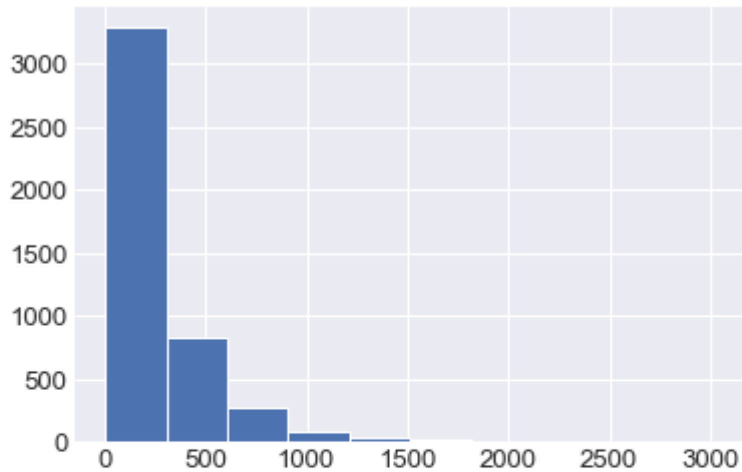
In [24]:
```
plt.hist(bank.balance)
```

Out[24]:
```
(array([4.111e+03, 3.400e+02, 4.700e+01, 1.700e+01, 4.000e+00, 0.000e+00,
        1.000e+00, 0.000e+00, 0.000e+00, 1.000e+00]),
 array([-3313. ,  4137.1, 11587.2, 19037.3, 26487.4, 33937.5, 41387.6,
        48837.7, 56287.8, 63737.9, 71188. ]),
 <a list of 10 Patch objects>)
```



In [25]:
```
plt.hist(bank.duration)
```

Out[25]:
```
(array([3.285e+03, 8.250e+02, 2.670e+02, 9.100e+01, 2.900e+01, 1.600e+01,
        5.000e+00, 0.000e+00, 1.000e+00, 2.000e+00]),
 array([   4. ,  306.1,  608.2,  910.3, 1212.4, 1514.5, 1816.6, 2118.7,
        2420.8, 2722.9, 3025. ]),
 <a list of 10 Patch objects>)
```



In [26]:
```
# standard scores for the columns
scaler = StandardScaler()
```

In [27]:
```
# the model data will be standardized form of preliminary model data
bank.age = scaler.fit_transform(bank.age.values.reshape(-1,1))
bank.balance = scaler.fit_transform(bank.balance.values.reshape(-1,1))
bank.duration = scaler.fit_transform(bank.duration.values.reshape(-1,1))
bank.campaign = scaler.fit_transform(bank.campaign.values.reshape(-1,1))
bank.pdays = scaler.fit_transform(bank.pdays.values.reshape(-1,1))
bank.previous = scaler.fit_transform(bank.previous.values.reshape(-1,1))
```

In [28]:
```
bank.head()
```

Out[28]:

	age	job	marital	education	default	balance	housing	loan	contact	day	mo
0	-1.056270	unemployed	married	primary	0	0.121072	0	0	cellular	19	
1	-0.772583	services	married	secondary	0	1.118644	1	1	cellular	11	r
2	-0.583458	management	single	tertiary	0	-0.024144	1	0	cellular	16	
3	-1.056270	management	married	tertiary	0	0.017726	1	1	unknown	3	
4	1.686036	blue-collar	married	secondary	0	-0.472753	1	0	unknown	5	r

In [29]:
```
bank['response'].value_counts()
```

Out[29]:
```
0    4000
1     521
Name: response, dtype: int64
```

In [30]:
```
bank.groupby('response').mean()
```

Out[30]:

	age	default	balance	housing	loan	day	duration	campaign	p
response									
0	-0.016274	0.016750	-0.006462	0.584750	0.162000	15.948750	-0.144764	0.022068	-0.0:
1	0.124942	0.017274	0.049612	0.422265	0.082534	15.658349	1.111434	-0.169430	0.2

In [31]:
```
bank['job'].value_counts()
```

Out[31]:
```
management       969
blue-collar      946
technician       768
admin.           478
services         417
retired          230
self-employed    183
entrepreneur     168
unemployed       128
housemaid        112
student           84
unknown           38
Name: job, dtype: int64
```

In [32]:
```
bank['marital'].value_counts()
```

Out[32]:
```
married     2797
single      1196
divorced     528
Name: marital, dtype: int64
```

In [33]:
```
bank['education'].value_counts()
```

Out[33]:
```
secondary    2306
tertiary     1350
primary       678
unknown       187
Name: education, dtype: int64
```

In [34]:
```
bank['default'].value_counts()
```

Out[34]:
```
0    4445
1      76
Name: default, dtype: int64
```

In [35]:
```
bank['housing'].value_counts()
```

Out[35]:
```
1    2559
0    1962
Name: housing, dtype: int64
```

In [36]:
```
bank['loan'].value_counts()
```

Out[36]:
```
0    3830
1     691
Name: loan, dtype: int64
```

In [37]:
```
bank['contact'].value_counts()
```

Out[37]:
```
cellular     2896
unknown      1324
telephone     301
Name: contact, dtype: int64
```

In [38]:
```
bank['month'].value_counts()
```

Out[38]:
```
may    1398
jul     706
aug     633
jun     531
nov     389
apr     293
feb     222
jan     148
oct      80
sep      52
mar      49
dec      20
Name: month, dtype: int64
```

In [39]:
```
bank['poutcome'].value_counts()
```

Out[39]:
```
unknown    3705
failure     490
other       197
success     129
Name: poutcome, dtype: int64
```

In [40]:
```
cat_vars=['job','marital', 'education', 'contact', 'month', 'poutcome']
for var in cat_vars:
    cat_list='var'+'_'+var
    cat_list = pd.get_dummies(bank[var], prefix=var)
    bank1=bank.join(cat_list)
    bank=bank1
```

In [41]:
```
bank_categorical = pd.DataFrame(data = bank, columns = ["response", "job",
```

In [42]:
```
bank_categorical
```

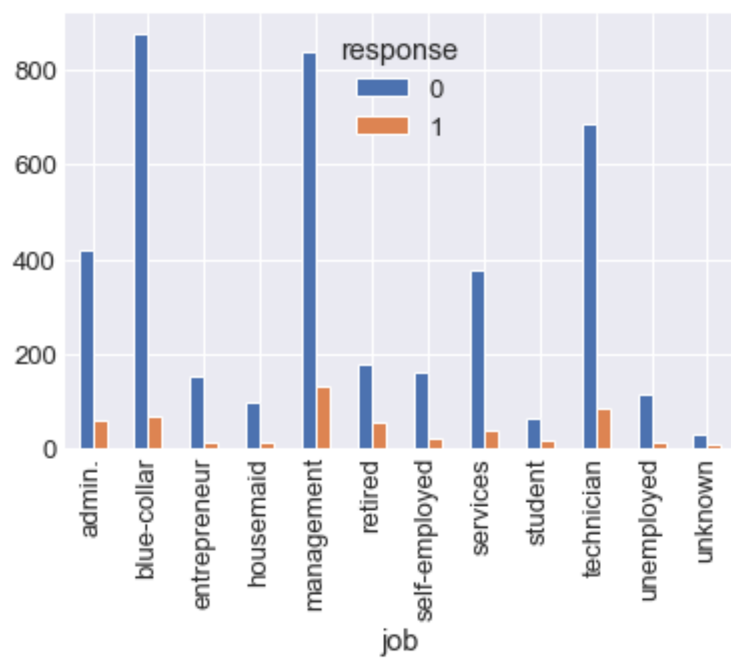
Out[42]:

	response	job	marital	education	contact	month	poutcome
0	0	unemployed	married	primary	cellular	oct	unknown
1	0	services	married	secondary	cellular	may	failure
2	0	management	single	tertiary	cellular	apr	failure
3	0	management	married	tertiary	unknown	jun	unknown
4	0	blue-collar	married	secondary	unknown	may	unknown
...	...	...	...	...	...	...	...
4516	0	services	married	secondary	cellular	jul	unknown
4517	0	self-employed	married	tertiary	unknown	may	unknown
4518	0	technician	married	secondary	cellular	aug	unknown
4519	0	blue-collar	married	secondary	cellular	feb	other
4520	0	entrepreneur	single	tertiary	cellular	apr	other

4521 rows × 7 columns

In [43]:
```
pd.crosstab(bank_categorical["job"], bank_categorical["response"]).plot(kin
```

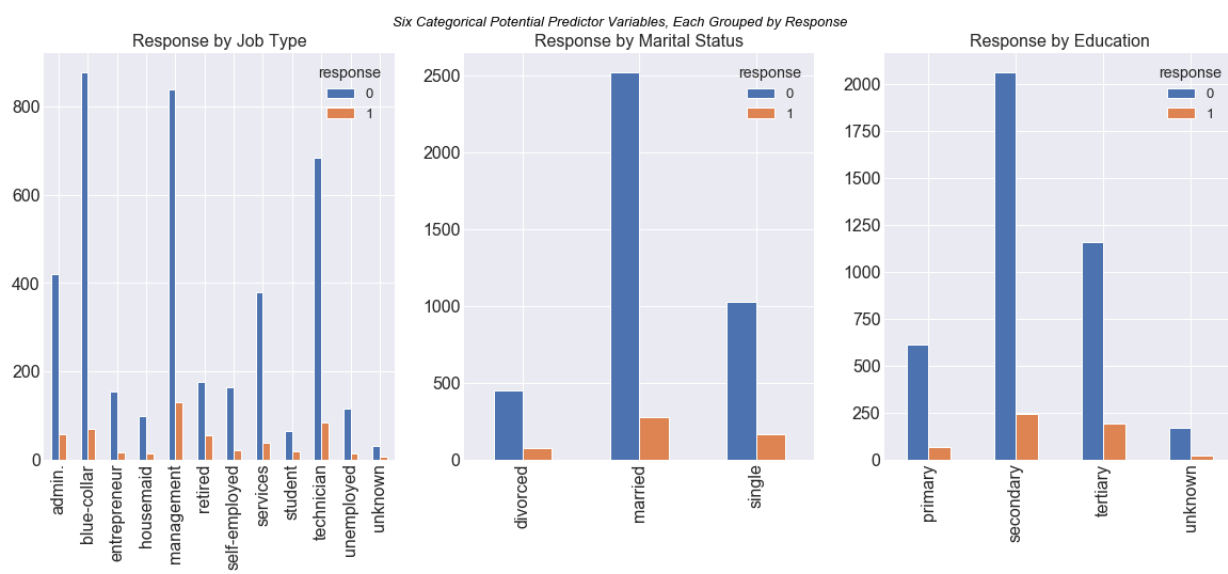
Out[43]: <matplotlib.axes._subplots.AxesSubplot at 0x12d575ad0>

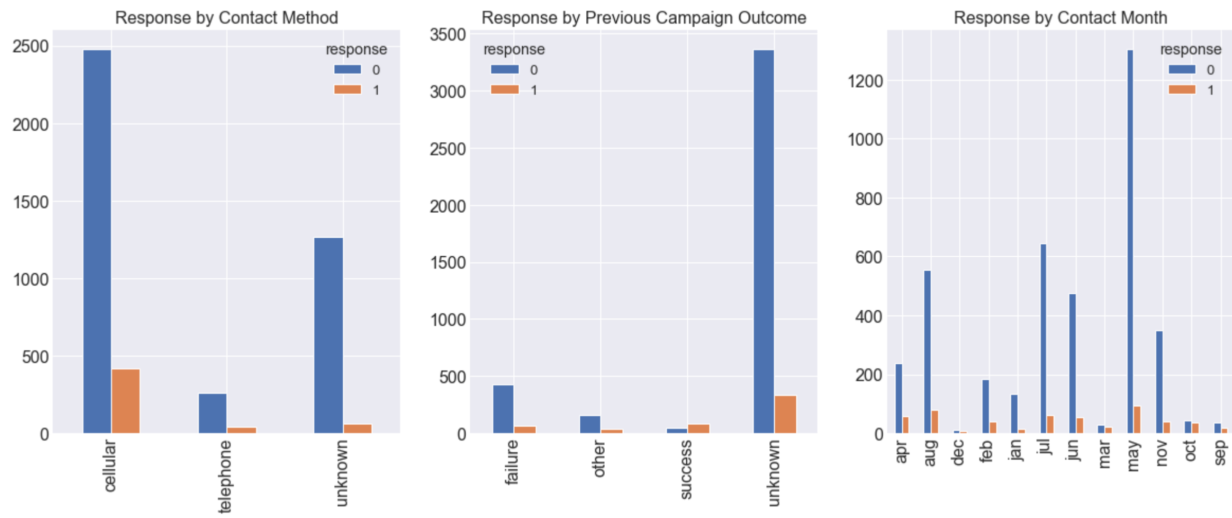


In [44]:

```
plt.style.use('seaborn-darkgrid')

fig1, (ax1, ax2, ax3) = plt.subplots(ncols=3, figsize=(20, 7))
pd.crosstab(bank_categorical["job"], bank_categorical["response"]).plot(kin
pd.crosstab(bank_categorical["marital"], bank_categorical["response"]).plot
pd.crosstab(bank_categorical["education"], bank_categorical["response"]).pl

# general title
plt.suptitle("Six Categorical Potential Predictor Variables, Each Grouped b



fig2, (ax4, ax5, ax6) = plt.subplots(ncols=3, figsize=(20, 7))
pd.crosstab(bank_categorical["contact"], bank_categorical["response"]).plot
pd.crosstab(bank_categorical["poutcome"], bank_categorical["response"]).plo
pd.crosstab(bank_categorical["month"], bank_categorical["response"]).plot(k

ax1.set_title("Response by Job Type", fontsize = 16)
ax1.xaxis.set_label_text("")
ax2.set_title("Response by Marital Status", fontsize = 16)
ax2.xaxis.set_label_text("")
ax3.set_title("Response by Education", fontsize = 16)
ax3.xaxis.set_label_text("")

ax4.set_title("Response by Contact Method", fontsize = 16)
ax4.xaxis.set_label_text("")
ax5.set_title("Response by Previous Campaign Outcome", fontsize = 16)
ax5.xaxis.set_label_text("")
ax6.set_title("Response by Contact Month", fontsize = 16)
ax6.xaxis.set_label_text("")
```

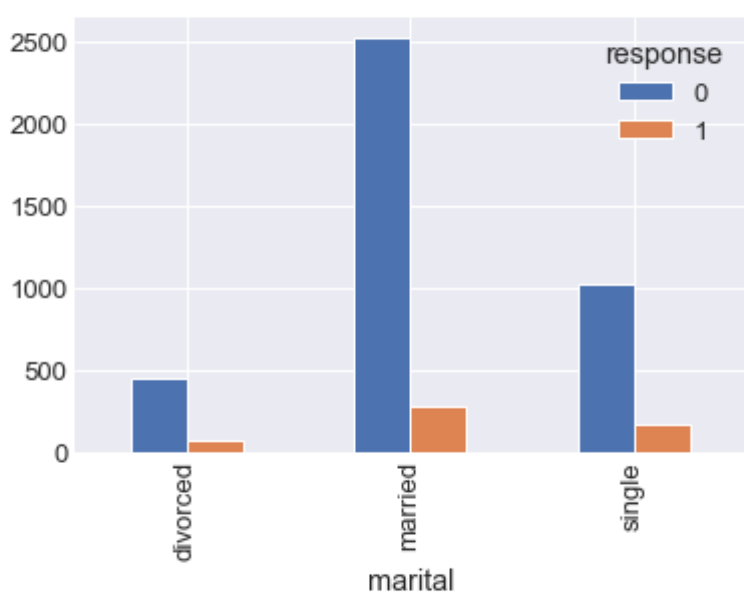
Out[44]: Text(0.5, 0, '')





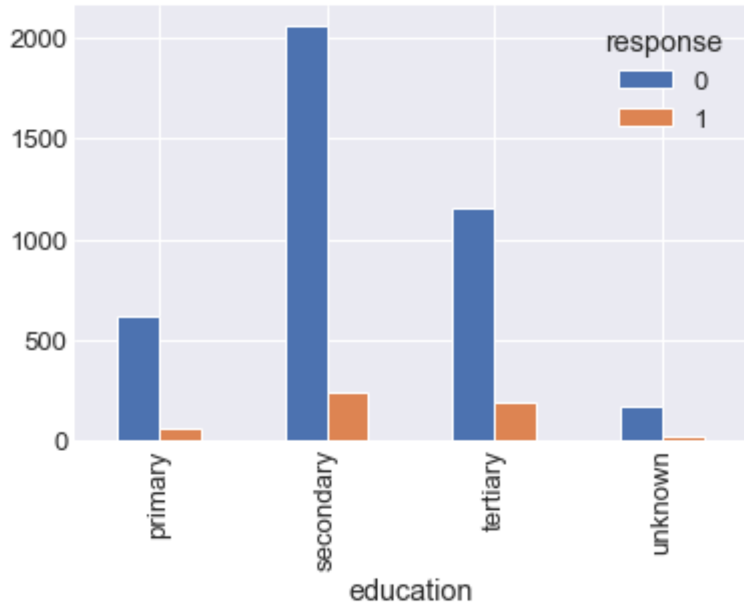
In [45]:
```
pd.crosstab(bank_categorical["marital"], bank_categorical["response"]).plot
```

Out[45]: <matplotlib.axes._subplots.AxesSubplot at 0x12c5d3950>



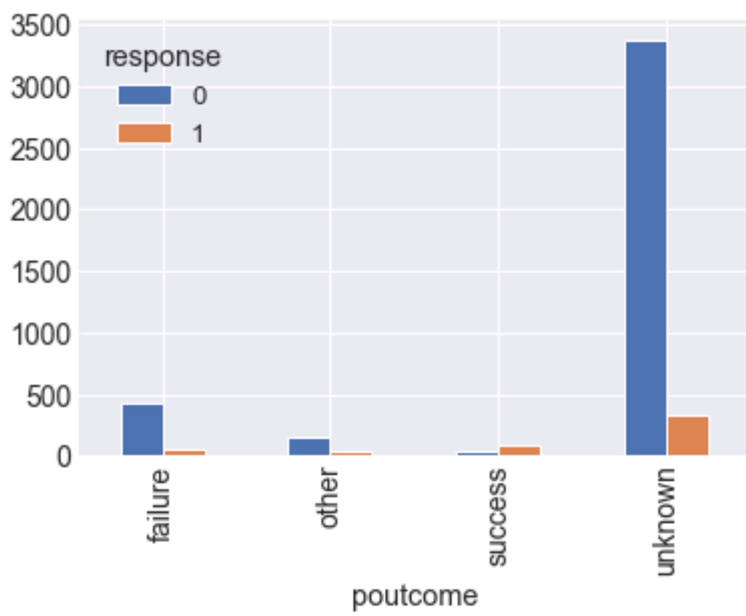
In [46]:
```
pd.crosstab(bank_categorical["education"], bank_categorical["response"]).pl
```

Out[46]: <matplotlib.axes._subplots.AxesSubplot at 0x12c9a2610>



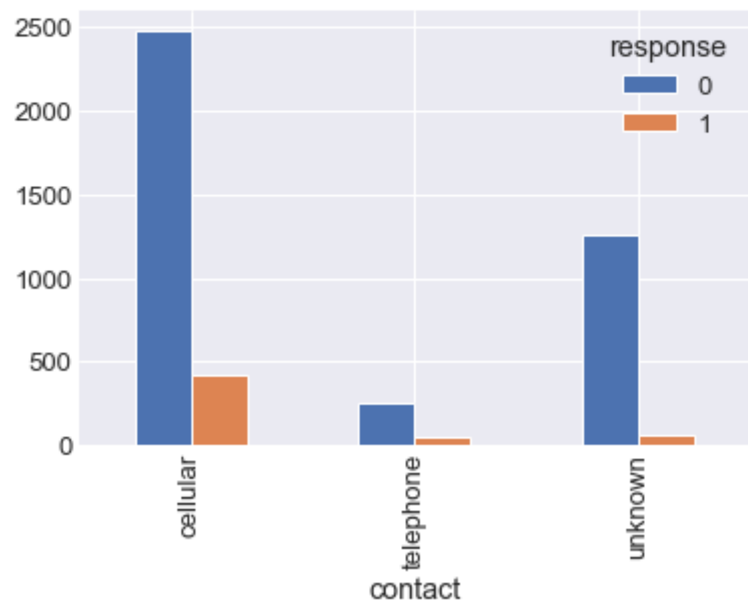
In [47]:
```
pd.crosstab(bank_categorical["poutcome"], bank_categorical["response"]).plo
```

Out[47]: <matplotlib.axes._subplots.AxesSubplot at 0x12cca5810>



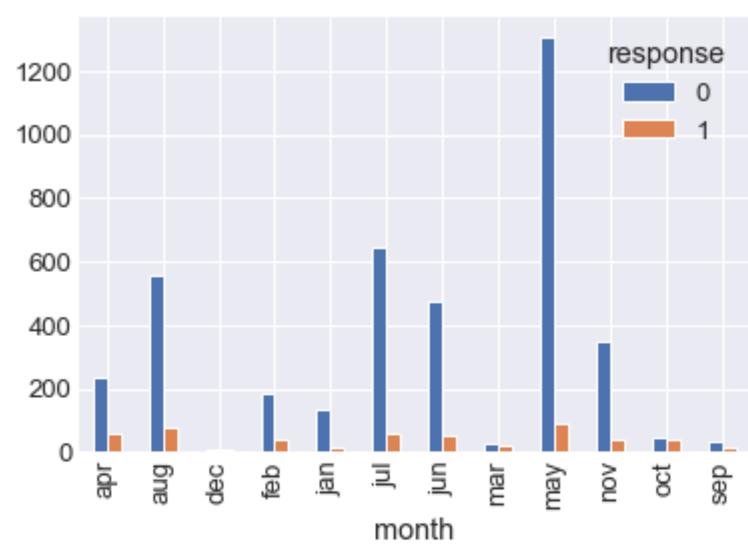
In [48]:
```
pd.crosstab(bank_categorical["contact"], bank_categorical["response"]).plot
```

Out[48]: <matplotlib.axes._subplots.AxesSubplot at 0x12cc73190>



In [49]:
```
pd.crosstab(bank_categorical["month"], bank_categorical["response"]).plot(k
```

Out[49]: <matplotlib.axes._subplots.AxesSubplot at 0x12d6bbfd0>



In [50]:
```
bank = bank.drop(['job','marital', 'education', 'contact', 'month', 'poutco
```

In [51]:
```
bank.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4521 entries, 0 to 4520
Data columns (total 49 columns):
age                    4521 non-null float64
default                4521 non-null int64
balance                4521 non-null float64
housing                4521 non-null int64
loan                   4521 non-null int64
day                    4521 non-null int64
duration               4521 non-null float64
campaign               4521 non-null float64
pdays                  4521 non-null float64
previous               4521 non-null float64
response               4521 non-null int64
job_admin.             4521 non-null uint8
job_blue-collar        4521 non-null uint8
job_entrepreneur       4521 non-null uint8
job_housemaid          4521 non-null uint8
job_management         4521 non-null uint8
job_retired            4521 non-null uint8
job_self-employed      4521 non-null uint8
job_services           4521 non-null uint8
job_student            4521 non-null uint8
job_technician         4521 non-null uint8
job_unemployed         4521 non-null uint8
job_unknown            4521 non-null uint8
marital_divorced       4521 non-null uint8
marital_married        4521 non-null uint8
marital_single         4521 non-null uint8
education_primary      4521 non-null uint8
education_secondary    4521 non-null uint8
education_tertiary     4521 non-null uint8
education_unknown      4521 non-null uint8
contact_cellular       4521 non-null uint8
contact_telephone      4521 non-null uint8
contact_unknown        4521 non-null uint8
month_apr              4521 non-null uint8
month_aug              4521 non-null uint8
month_dec              4521 non-null uint8
month_feb              4521 non-null uint8
month_jan              4521 non-null uint8
month_jul              4521 non-null uint8
month_jun              4521 non-null uint8
month_mar              4521 non-null uint8
month_may              4521 non-null uint8
month_nov              4521 non-null uint8
month_oct              4521 non-null uint8
month_sep              4521 non-null uint8
poutcome_failure       4521 non-null uint8
poutcome_other         4521 non-null uint8
poutcome_success       4521 non-null uint8
poutcome_unknown       4521 non-null uint8
dtypes: float64(6), int64(5), uint8(38)
memory usage: 556.4 KB
```

In [52]:
```
cor = bank.corr()
#Correlation with output variable
cor_target = abs(cor["response"])
#Selecting highly correlated features
relevant_features = cor_target[cor_target>0.10]
relevant_features
```

Out[52]:
```
housing             0.104683
duration            0.401118
pdays               0.104087
previous            0.116714
response            1.000000
contact_cellular    0.118761
contact_unknown     0.139399
month_mar           0.102716
month_may           0.102077
month_oct           0.145964
poutcome_success    0.283481
poutcome_unknown    0.162038
Name: response, dtype: float64
```

In [53]:
```
bank[[ "response", "default","housing", "loan"]].corr()
```

Out[53]:

	response	default	housing	loan
response	1.000000	0.001303	-0.104683	-0.070517
default	0.001303	1.000000	0.006881	0.063994
housing	-0.104683	0.006881	1.000000	0.018451
loan	-0.070517	0.063994	0.018451	1.000000

In [54]:
```
bank.shape
```

Out[54]:
```
(4521, 49)
```

Spliting Data

In [55]:
```
#from class lecture

X = pd.DataFrame(bank[["loan", "default", "housing", "duration", "previous"
trainnum = random.sample(range(1,4521), 900)
train = X.loc[trainnum]
test = X.drop(X.index[trainnum])

train_X = train[["loan", "default", "housing", "duration"]]
y_train = val.column_or_1d(train[["response"]])
print(np.mean(X["response"]))

X_test = np.array(test[["loan", "default", "housing", "duration", "previous
y_test = np.array(val.column_or_1d(test[["response"]]))
```

```
0.11523999115239991
```

Oversampling of Training

In [56]:
```
#from class lecture

minority = train[train["response"]==1]
majority = train[train["response"]==0]

newbank = resample(minority, replace = True, n_samples = len(majority), ran
newbank = pd.concat([majority, newbank])
newbank.response.value_counts()

X_train = np.array(newbank[["loan", "default", "housing", "duration", "prev
y_train = val.column_or_1d(newbank[["response"]])
```

In [ ]:

Logistic Regression on Training

In [57]:
```
#from class lecture

nfolds = 10
clf = lr(solver = "lbfgs", multi_class = "ovr")
mycvs = cvs(clf, X_train,y_train, cv=nfolds)
print("Accuracy of LR: ", mycvs)
```

```
Accuracy of LR:  [0.85625    0.7875     0.75       0.78125    0.79375
0.84375
 0.81875    0.79375    0.7721519  0.75316456]
```

In [58]:
```
statistics.mean(mycvs)
```

Out[58]:
```
0.7950316455696202
```

Native Bayes

In [59]:
```
clf1 = bern()
mycvs1 = cvs(clf1, X_train, y_train, cv=nfolds)
print("Accuracy of NB: ", mycvs1)
```

```
Accuracy of NB:  [0.79375    0.73125    0.71875    0.7125     0.775
0.8375
 0.80625    0.75       0.74050633 0.75949367]
```

In [60]:
```
statistics.mean(mycvs1)
```

Out[60]:
```
0.7625
```

Fit to the Full Training Set

In [61]:
```
clf.fit(X_train, y_train)
mypred = clf.predict_proba(X_train)
mypred = [p[1] for p in mypred]
mypredclass = clf.predict(X_train)

clf1.fit(X_train, y_train)
mypred1 = clf1.predict_proba(X_train)
mypred1 = [p[1] for p in mypred1]
mypredclass1 = clf1.predict(X_train)
```

In [62]:
```
clf
```

Out[62]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=Tr
ue,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='ovr', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose
=0,
                   warm_start=False)
```

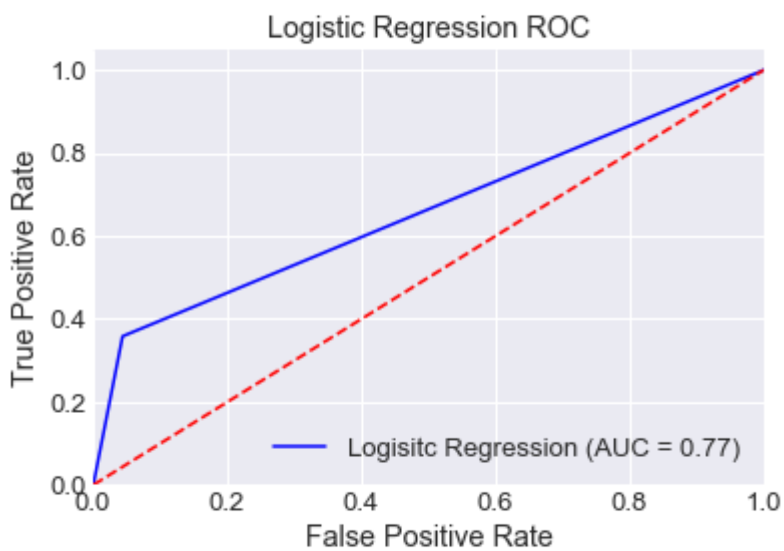
Fit on Test Set

In [63]:
```
mypred = clf.predict_proba(X_test)
mypred = [p[1] for p in mypred]
mypredclass = clf.predict(X_test)

mypred1 = clf1.predict_proba(X_test)
mypred1 = [p[1] for p in mypred1]
mypredclass1 = clf1.predict(X_test)
```

In [64]:
```
fpr, tpr, thresholds = roc_curve(mypredclass, y_test)
roc_auc = auc(fpr, tpr)
```

In [65]:
```
auc_lr = round(roc_auc_score(y_test, mypredclass),3)
```

In [66]:
```
fpr, tpr, thresholds = roc_curve(mypredclass, y_test)
roc_auc = auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, color = "blue", label='Logisitc Regression (AUC = %0.2f)
plt.plot([0,1], [0,1], color = "red", linestyle = "--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Logistic Regression ROC')
plt.legend(loc="lower right")
plt.show()
```



In [93]:
```
tn, fp, fn, tp = confusion_matrix(y_test, mypredclass).ravel()
print("Logistic Regression")
print("------------------------")
print(f'True Positives: {tp}')
print(f'False Positives: {fp}')
print(f'True Negatives: {tn}')
print(f'False Negatives: {fn}')
print("-------------------------")
print(f'True Positive Rate (Sensitivity): {round(tp /(tp + fn),3)}')
print(f'False Positives: {round(fp / (fp+tn),3)}')
print(f'True Negatives (Specificity): {round(tn / (fp + tn),3)}')
print(f'False Negatives: {round(fn / (fn + tp),3)}')
print("------------------------")
print(f"Area Under the Curve: {auc_lr}")
```

```
Logistic Regression
------------------------
True Positives: 297
False Positives: 532
True Negatives: 2670
False Negatives: 122
-------------------------
True Positive Rate (Sensitivity): 0.709
False Positives: 0.166
True Negatives (Specificity): 0.834
False Negatives: 0.291
------------------------
Area Under the Curve: 0.771
```

In [68]:
```
lr_intercept = clf.intercept_
lr_intercept
```

Out[68]:
```
array([-0.0204187])
```

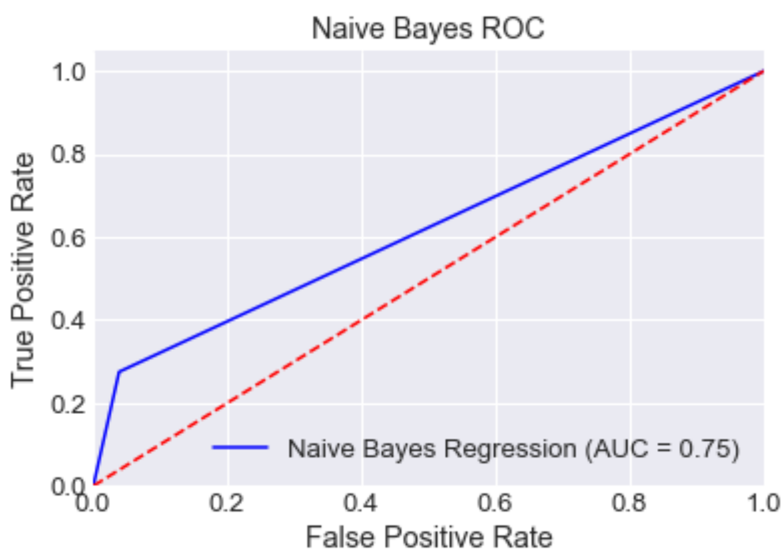
In [69]:
```
lr_coef = clf.coef_
lr_coef
```

Out[69]:
```
array([[-0.84581616,  0.65893661, -0.89442264,  1.17123347,  0.22276273,
         2.18865198]])
```

In [70]:
```
feat = ["loan", "default", "housing", "duration", "previous","poutcome_succ
```

In [71]:
```
auc_nb = round(roc_auc_score(y_test, mypredclass1),3)
```

In [72]:
```
fpr, tpr, thresholds = roc_curve(mypredclass1, y_test)
roc_auc = auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, color = "blue", label='Naive Bayes Regression (AUC = %0.
plt.plot([0,1], [0,1], color = "red", linestyle = "--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Naive Bayes ROC')
plt.legend(loc="lower right")
plt.show()
```



In [92]:
```
tn, fp, fn, tp = confusion_matrix(y_test, mypredclass1).ravel()
print("Naive Bayes")
print("------------------------")
print(f'True Positives: {tp}')
print(f'False Positives: {fp}')
print(f'True Negatives: {tn}')
print(f'False Negatives: {fn}')
print("------------------------")
print(f'True Positive Rate (Sensitivity): {round(tp /(tp + fn),3)}')
print(f'False Positives: {round(fp / (fp+tn),3)}')
print(f'True Negatives (Specificity): {round(tn / (fp + tn),3)}')
print(f'False Negatives: {round(fn / (fn + tp),3)}')
print("------------------------")
print(f"Area Under the Curve: {auc_nb}")
```

```
Naive Bayes
------------------------
True Positives: 325
False Positives: 857
True Negatives: 2345
False Negatives: 94
------------------------
True Positive Rate (Sensitivity): 0.776
False Positives: 0.268
True Negatives (Specificity): 0.732
False Negatives: 0.224
------------------------
Area Under the Curve: 0.754
```

In [74]:
```
nb_intercept = clf1.intercept_
nb_intercept
```

Out[74]:
```
array([-0.69314718])
```

In [75]:
```
nb_coef = clf1.coef_
nb_coef
```

Out[75]:
```
array([[-2.65926004, -4.28671645, -0.88855398, -0.31471074, -0.87347073,
        -1.72878467]])
```

In [76]:
```
# Naive Bayes confusion matrix
pd.DataFrame(confusion_matrix(y_test, mypredclass1), columns=['Predicted Re
```

Out[76]:

	Predicted Response No	Predicted Response Yes
Actual Response No	2345	857
Actual Response Yes	94	325

In [77]:
```
# Logisitic Regression confusion matrix
pd.DataFrame(confusion_matrix(y_test, mypredclass), columns=['Predicted Res
```

Out[77]:

	Predicted Response No	Predicted Response Yes
Actual Response No	2670	532
Actual Response Yes	122	297