In [2]:
```
# 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.score
# 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
```

In [3]:
```
# import base packages into the namespace for this program
import numpy as np
import pandas as pd
```

In [14]:
```
# initial work with the smaller data set
bank = pd.read_csv(r"C:\Users\Jimmy\Documents\bank.csv", sep = ';')  # start with s
maller data set
# examine the shape of original input data
print(bank.shape)
```

```
(4521, 17)
```

In [15]:
```
# 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 [16]:
```
# look at the list of column names, note that y is the response
list(bank.columns.values)
```

Out[16]:
```
['age',
 'job',
 'marital',
 'education',
 'default',
 'balance',
 'housing',
 'loan',
 'contact',
 'day',
 'month',
 'duration',
 'campaign',
 'pdays',
 'previous',
 'poutcome',
 'response']
```

In [17]:
```
# look at the beginning of the DataFrame
bank.head()
```

Out[17]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	can
0	30	unemployed	married	primary	no	1787	no	no	cellular	19	oct	79	
1	33	services	married	secondary	no	4789	yes	yes	cellular	11	may	220	
2	35	management	single	tertiary	no	1350	yes	no	cellular	16	apr	185	
3	30	management	married	tertiary	no	1476	yes	yes	unknown	3	jun	199	
4	59	blue-collar	married	secondary	no	0	yes	no	unknown	5	may	226	

In [18]:
```
#Let's look into dataset as a whole
# Plots and counts for important data that do not have numerical values
bank.describe()
```

Out[18]:

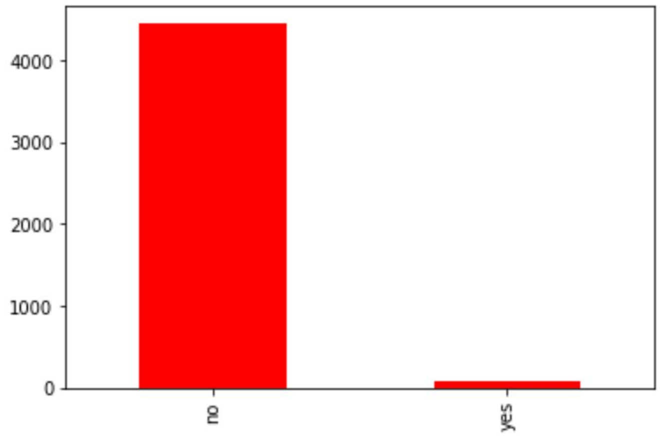
	age	balance	day	duration	campaign	pdays	previous
count	4521.000000	4521.000000	4521.000000	4521.000000	4521.000000	4521.000000	4521.000000
mean	41.170095	1422.657819	15.915284	263.961292	2.793630	39.766645	0.542579
std	10.576211	3009.638142	8.247667	259.856633	3.109807	100.121124	1.693562
min	19.000000	-3313.000000	1.000000	4.000000	1.000000	-1.000000	0.000000
25%	33.000000	69.000000	9.000000	104.000000	1.000000	-1.000000	0.000000
50%	39.000000	444.000000	16.000000	185.000000	2.000000	-1.000000	0.000000
75%	49.000000	1480.000000	21.000000	329.000000	3.000000	-1.000000	0.000000
max	87.000000	71188.000000	31.000000	3025.000000	50.000000	871.000000	25.000000

In [19]:
```
import matplotlib.pyplot as plt
import seaborn as sns
bank['response'].value_counts().plot(kind="bar")
plt.show()
```

```
<Figure size 640x480 with 1 Axes>
```

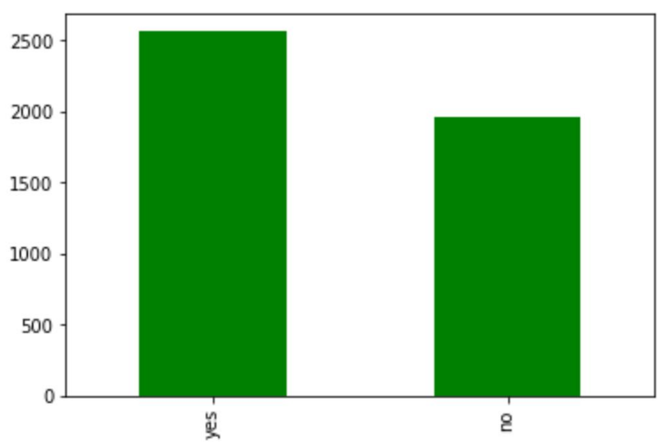
In [20]:
```
bank['response'].value_counts()
```
Out[20]:
```
no     4000
yes     521
Name: response, dtype: int64
```

In [21]:
```
bank['default'].value_counts().plot(kind="bar",color= 'red')
plt.show()
```



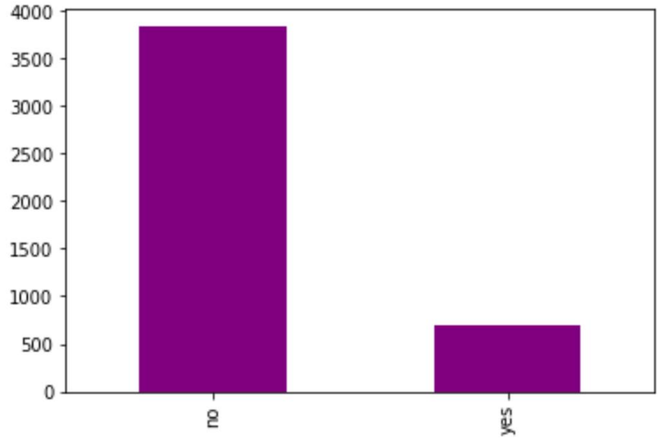
In [22]:
```
bank['default'].value_counts()
```
Out[22]:
```
no     4445
yes      76
Name: default, dtype: int64
```

In [23]:
```
bank['housing'].value_counts().plot(kind="bar",color= 'green')
plt.show()
```



In [24]:
```
bank['housing'].value_counts()
```
Out[24]:
```
yes    2559
no     1962
Name: housing, dtype: int64
```

In [25]:
```
bank['loan'].value_counts().plot(kind="bar",color= 'purple')
plt.show()
```



In [26]:
```
bank['loan'].value_counts()
```

Out[26]:
```
no     3830
yes     691
Name: loan, dtype: int64
```

In [27]:
```
print ("****Job distibution:")
print (bank['job'].value_counts())
print()
print ("****Marital status distibution:")
print (bank['marital'].value_counts())
print()
print ("****Education distibution :")
print (bank['education'].value_counts())
print()
print ("****Default distibution:")
print (bank['default'].value_counts())
print()
print ("****Housing distibution :")
print (bank['housing'].value_counts())
print()
print ("****Loan distibution:")
print (bank['loan'].value_counts())
print()
print ("****Contact distibution:")
print (bank['contact'].value_counts())
print()
print ("****Month distibution :")
print (bank['month'].value_counts())
print()
```

```
****Job distibution:
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

****Marital status distibution:
married     2797
single      1196
divorced     528
Name: marital, dtype: int64

****Education distibution :
secondary    2306
tertiary     1350
primary       678
unknown       187
Name: education, dtype: int64

****Default distibution:
no     4445
yes      76
Name: default, dtype: int64

****Housing distibution :
yes    2559
no     1962
Name: housing, dtype: int64

****Loan distibution:
no     3830
yes     691
Name: loan, dtype: int64

****Contact distibution:
cellular     2896
unknown      1324
telephone     301
Name: contact, dtype: int64

****Month distibution :
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 [28]:
```
#Let's look specifically into who responded
response_yes=bank[(bank.response == "yes")]
response_yes.head()
```

Out[28]:

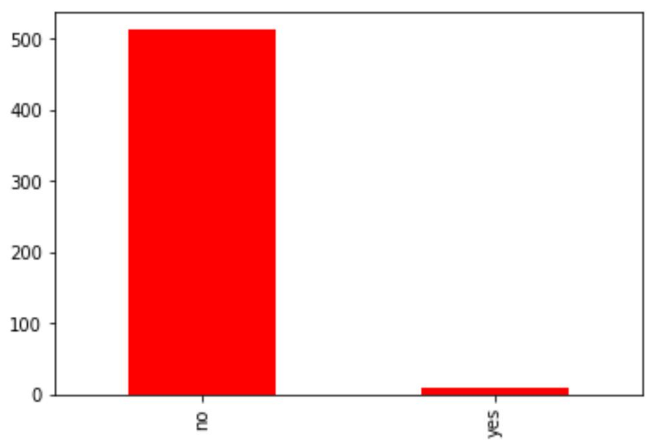
	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	c
13	20	student	single	secondary	no	502	no	no	cellular	30	apr	261	
30	68	retired	divorced	secondary	no	4189	no	no	telephone	14	jul	897	
33	32	management	single	tertiary	no	2536	yes	no	cellular	26	aug	958	
34	49	technician	married	tertiary	no	1235	no	no	cellular	13	aug	354	
36	78	retired	divorced	primary	no	229	no	no	telephone	22	oct	97	

In [29]:
```
response_yes.describe()
```

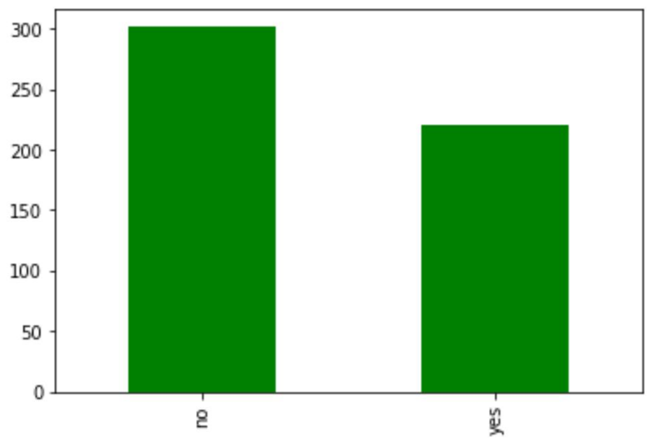
Out[29]:

	age	balance	day	duration	campaign	pdays	previous
count	521.000000	521.000000	521.000000	521.000000	521.000000	521.000000	521.000000
mean	42.491363	1571.955854	15.658349	552.742802	2.266795	68.639155	1.090211
std	13.115772	2444.398956	8.235148	390.325805	2.092071	121.963063	2.055368
min	19.000000	-1206.000000	1.000000	30.000000	1.000000	-1.000000	0.000000
25%	32.000000	171.000000	9.000000	260.000000	1.000000	-1.000000	0.000000
50%	40.000000	710.000000	15.000000	442.000000	2.000000	-1.000000	0.000000
75%	50.000000	2160.000000	22.000000	755.000000	3.000000	98.000000	2.000000
max	87.000000	26965.000000	31.000000	2769.000000	24.000000	804.000000	14.000000

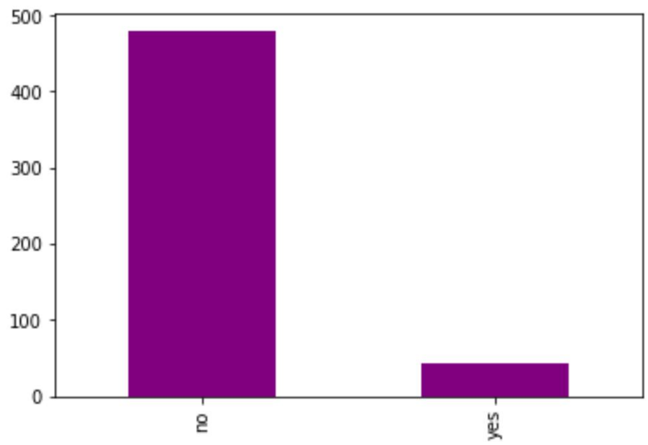
In [30]:
```
response_yes['default'].value_counts().plot(kind="bar",color= 'red')
plt.show()
```



In [31]:
```
response_yes['housing'].value_counts().plot(kind="bar",color= 'green')
plt.show()
```



In [32]:
```
response_yes['loan'].value_counts().plot(kind="bar",color= 'purple')
plt.show()
```



In [33]:
```
print ("****Job distibution for those who responded:")
print (response_yes['job'].value_counts())
print()
print ("****Marital status distibution for those who responded:")
print (response_yes['marital'].value_counts())
print()
print ("****Education distibution for those who responded:")
print (response_yes['education'].value_counts())
print()
print ("****Default distibution for those who responded:")
print (response_yes['default'].value_counts())
print()
print ("****Housing distibution for those who responded:")
print (response_yes['housing'].value_counts())
print()
print ("****Loan distibution for those who responded:")
print (response_yes['loan'].value_counts())
print()
print ("****Contact distibution for those who responded:")
print (response_yes['contact'].value_counts())
print()
print ("****Month distibution for those who responded:")
print (response_yes['month'].value_counts())
print()
```

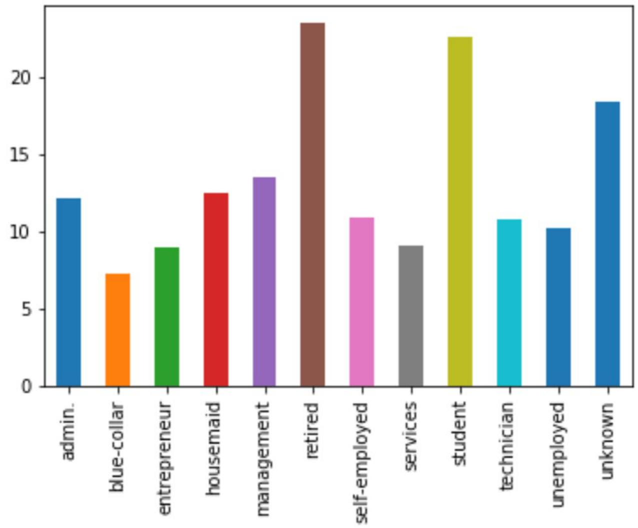
```
****Job distibution for those who responded:
management       131
technician        83
blue-collar       69
admin.            58
retired           54
services          38
self-employed     20
student           19
entrepreneur      15
housemaid         14
unemployed        13
unknown            7
Name: job, dtype: int64

****Marital status distibution for those who responded:
married     277
single      167
divorced     77
Name: marital, dtype: int64

****Education distibution for those who responded:
secondary    245
tertiary     193
primary       64
unknown       19
Name: education, dtype: int64

****Default distibution for those who responded:
no     512
yes      9
Name: default, dtype: int64

****Housing distibution for those who responded:
no     301
yes    220
Name: housing, dtype: int64

****Loan distibution for those who responded:
no     478
yes     43
Name: loan, dtype: int64

****Contact distibution for those who responded:
cellular     416
unknown       61
telephone     44
Name: contact, dtype: int64

****Month distibution for those who responded:
may    93
aug    79
jul    61
apr    56
jun    55
nov    39
feb    38
oct    37
mar    21
sep    17
jan    16
dec     9
Name: month, dtype: int64
```

In [34]:
```
#Percentage of each attribute for those who responded.  Total amount used for calcu
lation.  Should not add up to 100%
#Completed this way bacuase we already have a distribution above.
print ("****Job percentage for those who responded:")
print ((response_yes['job'].value_counts()/bank['job'].value_counts())*100)
print()
print ("****Marital status percentage for those who responded:")
print ((response_yes['marital'].value_counts()/bank['marital'].value_counts())*100)
print()
print ("****Education percentage for those who responded:")
print ((response_yes['education'].value_counts()/bank['education'].value_counts())*
100)
print()
print ("****Default percentage for those who responded:")
print ((response_yes['default'].value_counts()/bank['default'].value_counts())*100)
print()
print ("****Housing percentage for those who responded:")
print ((response_yes['housing'].value_counts()/bank['housing'].value_counts())*100)
print()
print ("****Loan percentage for those who responded:")
print ((response_yes['loan'].value_counts()/bank['loan'].value_counts())*100)
print()
print ("****Contact percentage for those who responded:")
print ((response_yes['contact'].value_counts()/bank['contact'].value_counts())*100)
print()
print ("****Month percentage for those who responded:")
print ((response_yes['month'].value_counts()/bank['month'].value_counts())*100)
print()
```

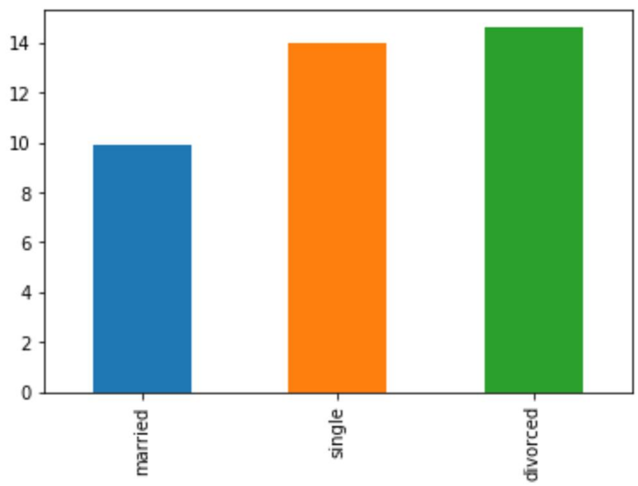
```
****Job percentage for those who responded:
admin.           12.133891
blue-collar       7.293869
entrepreneur      8.928571
housemaid        12.500000
management       13.519092
retired          23.478261
self-employed    10.928962
services          9.112710
student          22.619048
technician       10.807292
unemployed       10.156250
unknown          18.421053
Name: job, dtype: float64

****Marital status percentage for those who responded:
married      9.903468
single      13.963211
divorced    14.583333
Name: marital, dtype: float64

****Education percentage for those who responded:
secondary    10.624458
tertiary     14.296296
primary       9.439528
unknown      10.160428
Name: education, dtype: float64

****Default percentage for those who responded:
no     11.518560
yes    11.842105
Name: default, dtype: float64

****Housing percentage for those who responded:
no     15.341488
yes     8.597108
Name: housing, dtype: float64

****Loan percentage for those who responded:
no     12.480418
yes     6.222865
Name: loan, dtype: float64

****Contact percentage for those who responded:
cellular     14.364641
unknown       4.607251
telephone    14.617940
Name: contact, dtype: float64

****Month percentage for those who responded:
apr    19.112628
aug    12.480253
dec    45.000000
feb    17.117117
jan    10.810811
jul     8.640227
jun    10.357815
mar    42.857143
may     6.652361
nov    10.025707
oct    46.250000
sep    32.692308
Name: month, dtype: float64
```

In [195]:
```
#Percentage of each attribute for those who responded.  Total amount used for calc
ulation.  Should not add up to 100%
#Completed this way bacuase we already have a distribution above.
print ("****Job percentage for those who responded:")
((response_yes['job'].value_counts()/bank['job'].value_counts())*100).plot(kind='b
ar')
plt.show()
print()
print ("****Marital status percentage for those who responded:")
(((response_yes['marital'].value_counts()/bank['marital'].value_counts())*100)).pl
ot(kind='bar')
plt.show()
print()
print ("****Education percentage for those who responded:")
(((response_yes['education'].value_counts()/bank['education'].value_counts())*10
0)).plot(kind='bar')
plt.show()
print()
print ("****Default percentage for those who responded:")
(((response_yes['default'].value_counts()/bank['default'].value_counts())*100)).pl
ot(kind='bar')
plt.show()
print()
print ("****Housing percentage for those who responded:")
((response_yes['housing'].value_counts()/bank['housing'].value_counts())*100).plot
(kind='bar')
plt.show()
print()
print ("****Loan percentage for those who responded:")
((response_yes['loan'].value_counts()/bank['loan'].value_counts())*100).plot(kind=
'bar')
plt.show()
print()
print ("****Contact percentage for those who responded:")
((response_yes['contact'].value_counts()/bank['contact'].value_counts())*100).plot
(kind='bar')
plt.show()
print()
print ("****Month percentage for those who responded:")
((response_yes['month'].value_counts()/bank['month'].value_counts())*100).plot(kin
d='bar')
plt.show()
print()
```

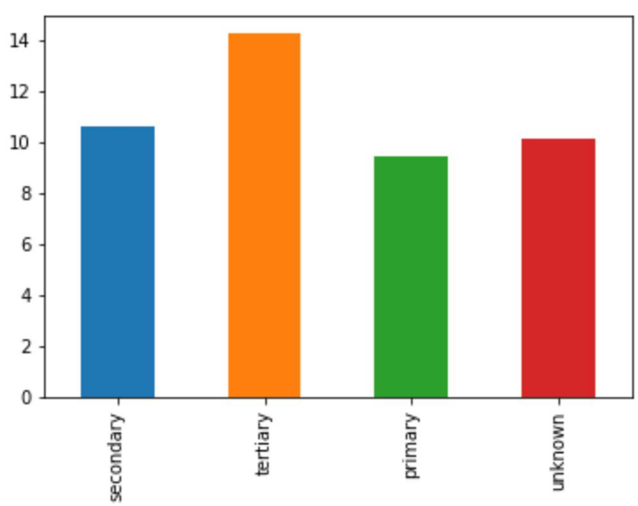
****Job percentage for those who responded:



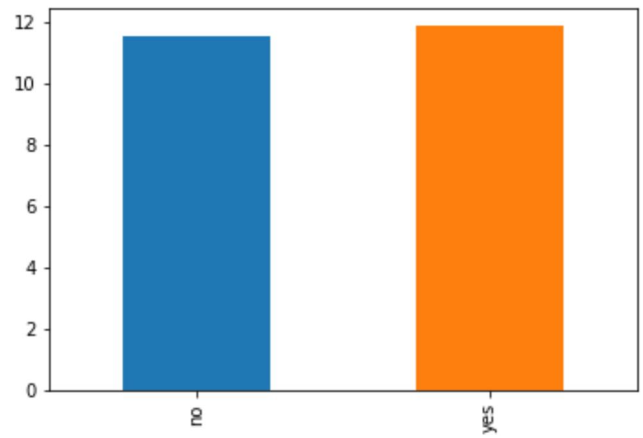
****Marital status percentage for those who responded:



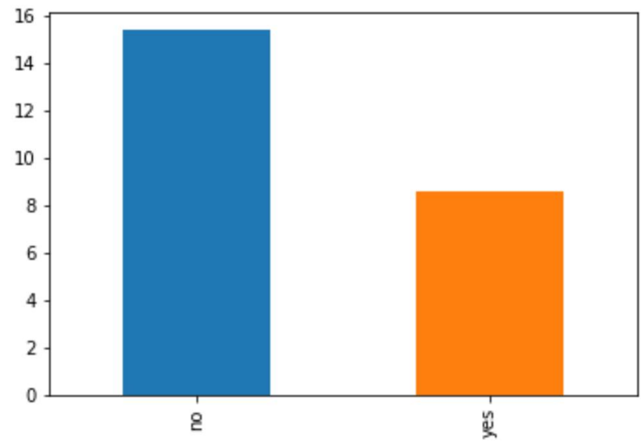
****Education percentage for those who responded:



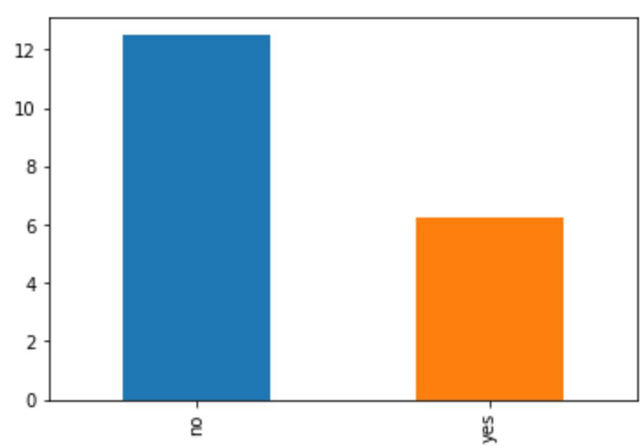
****Default percentage for those who responded:



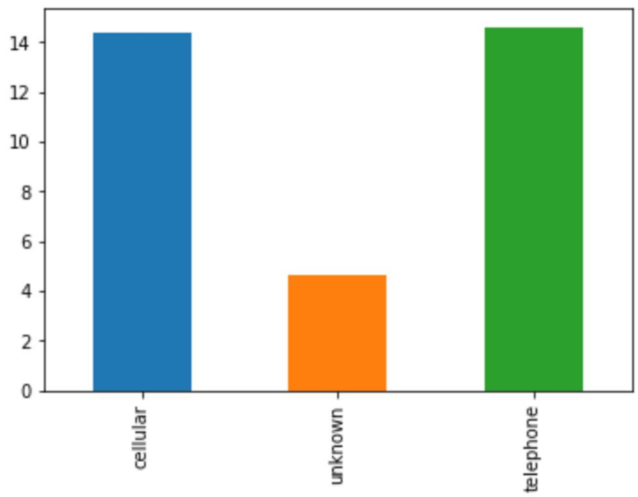

```
****Housing percentage for those who responded:
```



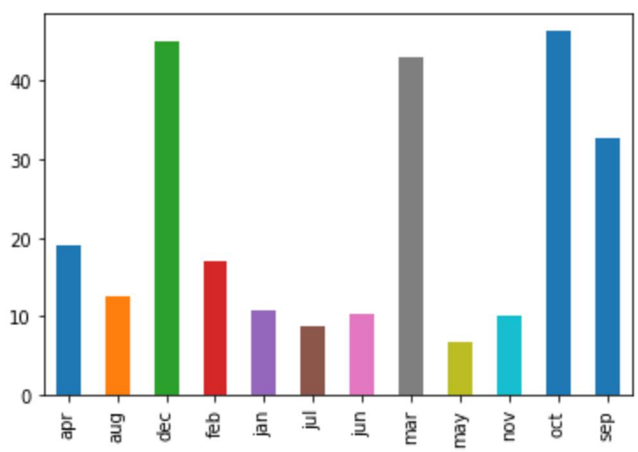

```
****Loan percentage for those who responded:
```




```
****Contact percentage for those who responded:
```



****Month percentage for those who responded:



In [189]:
```
response_no=bank[(bank.response == "no")]
response_no.head()
```

Out[189]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	can
0	30	unemployed	married	primary	no	1787	no	no	cellular	19	oct	79	
1	33	services	married	secondary	no	4789	yes	yes	cellular	11	may	220	
2	35	management	single	tertiary	no	1350	yes	no	cellular	16	apr	185	
3	30	management	married	tertiary	no	1476	yes	yes	unknown	3	jun	199	
4	59	blue-collar	married	secondary	no	0	yes	no	unknown	5	may	226	

In [190]:
```
response_no.describe()
```

Out[190]:

	age	balance	day	duration	campaign	pdays	previous
count	4000.000000	4000.000000	4000.000000	4000.000000	4000.000000	4000.000000	4000.000000
mean	40.998000	1403.211750	15.948750	226.347500	2.862250	36.006000	0.471250
std	10.188398	3075.349313	8.249736	210.313631	3.212609	96.297657	1.627371
min	19.000000	-3313.000000	1.000000	4.000000	1.000000	-1.000000	0.000000
25%	33.000000	61.000000	8.000000	96.000000	1.000000	-1.000000	0.000000
50%	39.000000	419.500000	16.000000	167.000000	2.000000	-1.000000	0.000000
75%	48.000000	1407.000000	21.000000	283.000000	3.000000	-1.000000	0.000000
max	86.000000	71188.000000	31.000000	3025.000000	50.000000	871.000000	25.000000

In [193]:
```
print ("****Job distibution for those who said no:")
print (response_no['job'].value_counts())
print()
print ("****Marital status distibution for those who said no:")
print (response_no['marital'].value_counts())
print()
print ("****Education distibution for those who said no:")
print (response_no['education'].value_counts())
print()
print ("****Default distibution for those who said no:")
print (response_no['default'].value_counts())
print()
print ("****Housing distibution for those who said no:")
print (response_no['housing'].value_counts())
print()
print ("****Loan distibution for those who said no:")
print (response_no['loan'].value_counts())
print()
print ("****Contact distibution for those who said no:")
print (response_no['contact'].value_counts())
print()
print ("****Month distibution for those who said no:")
print (response_no['month'].value_counts())
print()
```

```
****Job distibution for those who said no:
blue-collar      877
management       838
technician       685
admin.           420
services         379
retired          176
self-employed    163
entrepreneur     153
unemployed       115
housemaid         98
student           65
unknown           31
Name: job, dtype: int64

****Marital status distibution for those who said no:
married     2520
single      1029
divorced     451
Name: marital, dtype: int64

****Education distibution for those who said no:
secondary    2061
tertiary     1157
primary       614
unknown       168
Name: education, dtype: int64

****Default distibution for those who said no:
no     3933
yes      67
Name: default, dtype: int64

****Housing distibution for those who said no:
yes    2339
no     1661
Name: housing, dtype: int64

****Loan distibution for those who said no:
no     3352
yes     648
Name: loan, dtype: int64

****Contact distibution for those who said no:
cellular     2480
unknown      1263
telephone     257
Name: contact, dtype: int64

****Month distibution for those who said no:
may    1305
jul     645
aug     554
jun     476
nov     350
apr     237
feb     184
jan     132
oct      43
sep      35
mar      28
dec      11
Name: month, dtype: int64
```

In [194]:
```
#Percentage of each attribute for those who responded.  Total amount used for calc
ulation.  Should not add up to 100%
#Completed this way bacuase we already have a distribution above.
print ("****Job percentage for those who said no:")
print ((response_no['job'].value_counts()/bank['job'].value_counts())*100)
print()
print ("****Marital status percentage for those who said no:")
print ((response_no['marital'].value_counts()/bank['marital'].value_counts())*100)
print()
print ("****Education percentage for those who said no:")
print ((response_no['education'].value_counts()/bank['education'].value_counts())*
100)
print()
print ("****Default percentage for those who said no:")
print ((response_no['default'].value_counts()/bank['default'].value_counts())*100)
print()
print ("****Housing percentage for those who said no:")
print ((response_no['housing'].value_counts()/bank['housing'].value_counts())*100)
print()
print ("****Loan percentage for those who said no:")
print ((response_no['loan'].value_counts()/bank['loan'].value_counts())*100)
print()
print ("****Contact percentage for those who said no:")
print ((response_no['contact'].value_counts()/bank['contact'].value_counts())*100)
print()
print ("****Month percentage for those who said no:")
print ((response_no['month'].value_counts()/bank['month'].value_counts())*100)
print()
```

```
****Job percentage for those who said no:
admin.           87.866109
blue-collar      92.706131
entrepreneur     91.071429
housemaid        87.500000
management       86.480908
retired          76.521739
self-employed    89.071038
services         90.887290
student          77.380952
technician       89.192708
unemployed       89.843750
unknown          81.578947
Name: job, dtype: float64

****Marital status percentage for those who said no:
married     90.096532
single      86.036789
divorced    85.416667
Name: marital, dtype: float64

****Education percentage for those who said no:
secondary    89.375542
tertiary     85.703704
primary      90.560472
unknown      89.839572
Name: education, dtype: float64

****Default percentage for those who said no:
no     88.481440
yes    88.157895
Name: default, dtype: float64

****Housing percentage for those who said no:
yes    91.402892
no     84.658512
Name: housing, dtype: float64

****Loan percentage for those who said no:
no     87.519582
yes    93.777135
Name: loan, dtype: float64

****Contact percentage for those who said no:
cellular     85.635359
unknown      95.392749
telephone    85.382060
Name: contact, dtype: float64

****Month percentage for those who said no:
may    93.347639
jul    91.359773
aug    87.519747
jun    89.642185
nov    89.974293
apr    80.887372
feb    82.882883
jan    89.189189
oct    53.750000
sep    67.307692
mar    57.142857
dec    55.000000
Name: month, dtype: float64
```

In [35]:
```
# mapping function to convert text no/yes to integer 0/1
convert_to_binary = {'no' : 0, 'yes' : 1}
```

In [36]:
```
# define binary variable for having credit in default
default = bank['default'].map(convert_to_binary)
```

In [37]:
```
# define binary variable for having a mortgage or housing loan
housing = bank['housing'].map(convert_to_binary)
```

In [38]:
```
# define binary variable for having a personal loan
loan = bank['loan'].map(convert_to_binary)
```

In [39]:
```
# define response variable to use in the model
response = bank['response'].map(convert_to_binary)
```

In [165]:
```
# 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 [166]:
```
# examine the shape of model_data, which we will use in subsequent modeling
print(model_data.shape)
```

```
(4521, 4)
```

In [174]:
```
#Let's start modeling process
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.naive_bayes import BernoulliNB
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold

model_names = [ "Logistic_Regression","Naive_Bayes"]

models = [LogisticRegression(),BernoulliNB()]
```

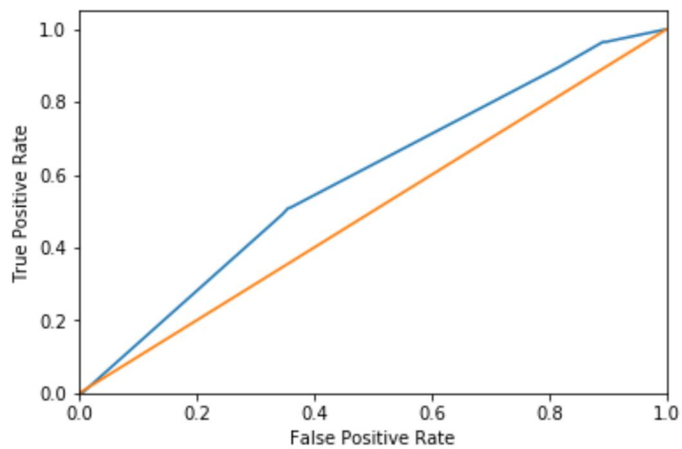
In [175]:
```
#Let's use kfold to split data.
#1:10 ratio for test:train data
N_FOLDS = 10
```

In [176]:
```
#will use to store
data_store = np.zeros((N_FOLDS, len(model_names)))
```

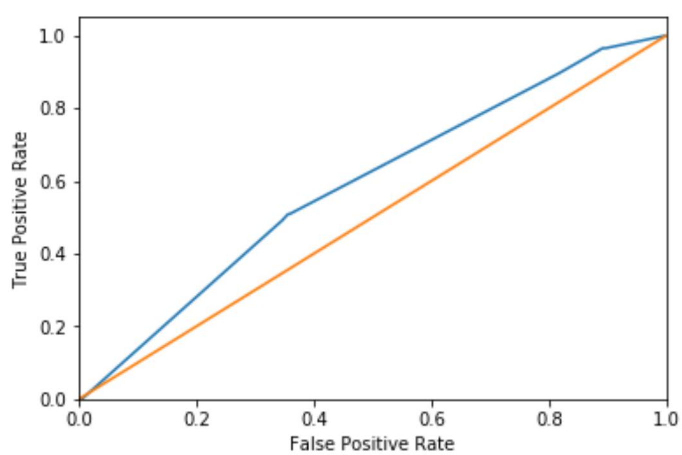
In [183]:

```
#Will use kfold to parse data
#In every 10 data points, 1 will be placed in test data set

kf = KFold(n_splits = N_FOLDS, shuffle=False, random_state = RANDOM_SEED)

index_for_fold = 0
for train_index, test_index in kf.split(model_data):
    print('Fold:', index_for_fold)
    print()
    X_train = model_data[train_index, 0:model_data.shape[1]-1]
    X_test = model_data[test_index, 0:model_data.shape[1]-1]
    y_train = model_data[train_index, model_data.shape[1]-1]
    y_test = model_data[test_index, model_data.shape[1]-1]

    print('Number of Data points and Variables:')
    print('X_train:', X_train.shape)
    print('X_test:',X_test.shape)
    print('y_train:', y_train.shape)
    print('y_test:',y_test.shape)
    print()

 #once separated into trainng and test datasets, let's find area under ROC curve f
or each model
    index_for_method = 0
    for name, cm in zip(model_names, models):
        print('Model:', name)
        cm.fit(X_train, y_train)
        y_test_predict = cm.predict_proba(X_test)
        roc_method_result = roc_auc_score(y_test, y_test_predict[:,1])
        print('Area under ROC curve:', roc_method_result)
        print()
        print()
        data_store[index_for_fold, index_for_method] = roc_method_result
        #Let's plot the ROC curve
        fpr, tpr, thresholds = roc_curve(y_test, y_test_predict[:,1])
        plt.figure()
        plt.plot(fpr, tpr)
        plt.plot([0, 1], [0, 1])
        plt.xlim([0.0, 1.0])
        plt.ylim([0.0, 1.05])
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.show()
        index_for_method += 1

    index_for_fold += 1

data_store_df = pd.DataFrame(data_store)
data_store_df.columns = model_names
print('********************************')
print('Average from 10 folds')
print('Method                 Area under ROC Curve')
print(data_store_df.mean())
print()
print('Standard Deviation')
print(data_store_df.std())
```

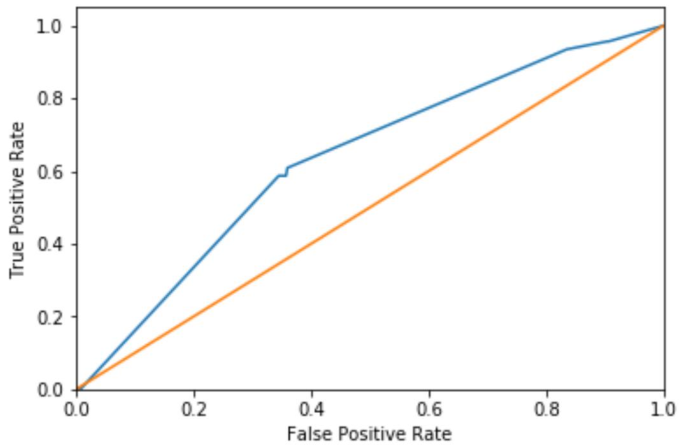
```
Fold: 0

Number of Data points and Variables:
X_train: (4068, 3)
X_test: (453, 3)
y_train: (4068,)
y_test: (453,)

Model: Logistic_Regression
Area under ROC curve: 0.5878522062732588
```



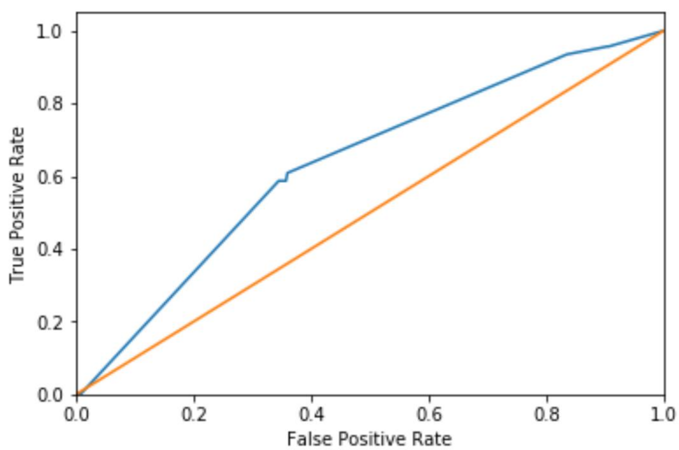

```
Model: Naive_Bayes
Area under ROC curve: 0.5878522062732588
```



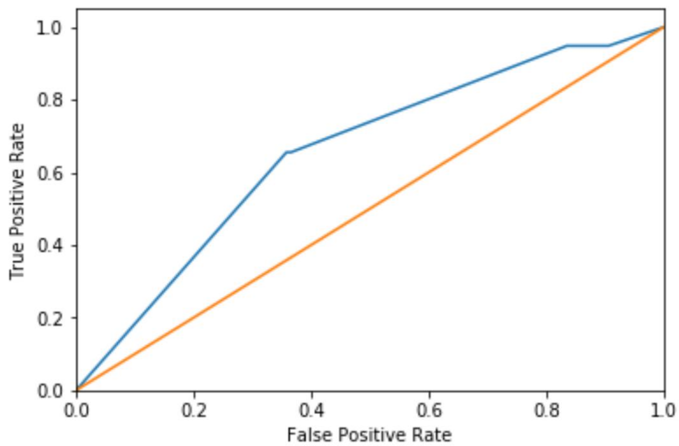

```
Fold: 1

Number of Data points and Variables:
X_train: (4069, 3)
X_test: (452, 3)
y_train: (4069,)
y_test: (452,)

Model: Logistic_Regression
Area under ROC curve: 0.633727778967659
```



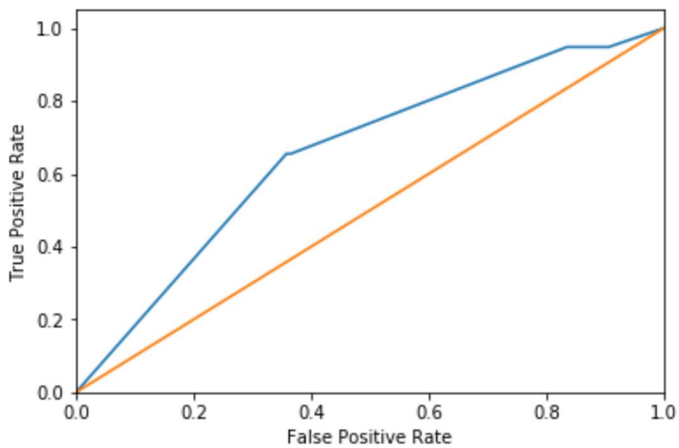

```
Model: Naive_Bayes
Area under ROC curve: 0.633727778967659
```



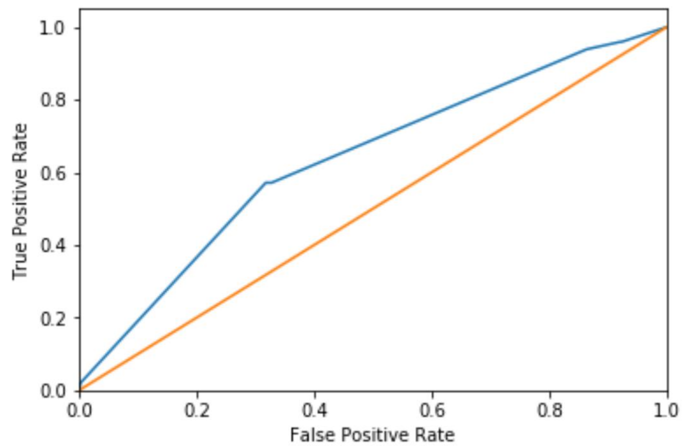

```
Fold: 2

Number of Data points and Variables:
X_train: (4069, 3)
X_test: (452, 3)
y_train: (4069,)
y_test: (452,)

Model: Logistic_Regression
Area under ROC curve: 0.6575354454752319
```



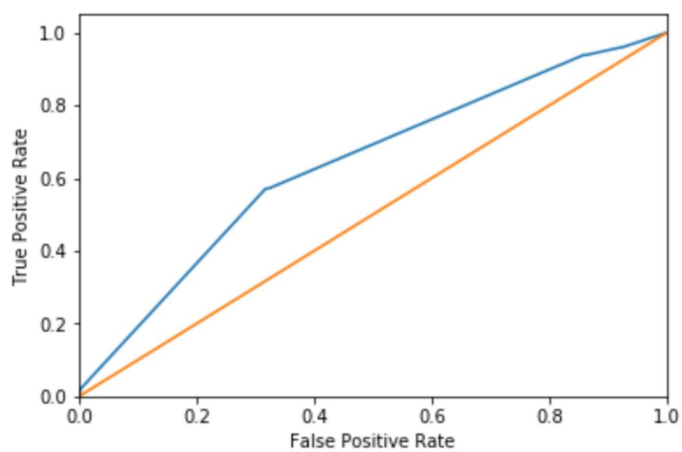

```
Model: Naive_Bayes
Area under ROC curve: 0.6575354454752319
```



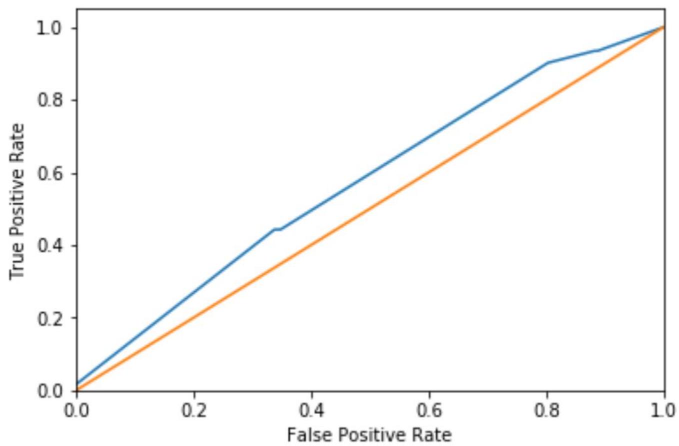

```
Fold: 3

Number of Data points and Variables:
X_train: (4069, 3)
X_test: (452, 3)
y_train: (4069,)
y_test: (452,)

Model: Logistic_Regression
Area under ROC curve: 0.6355648959335594
```



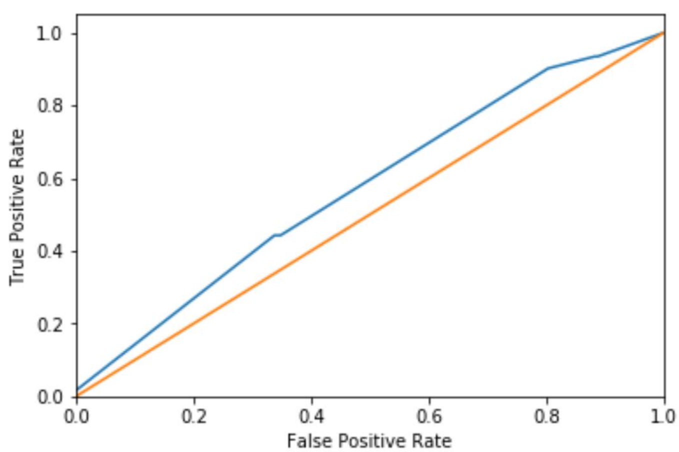

```
Model: Naive_Bayes
Area under ROC curve: 0.6373879576644552
```



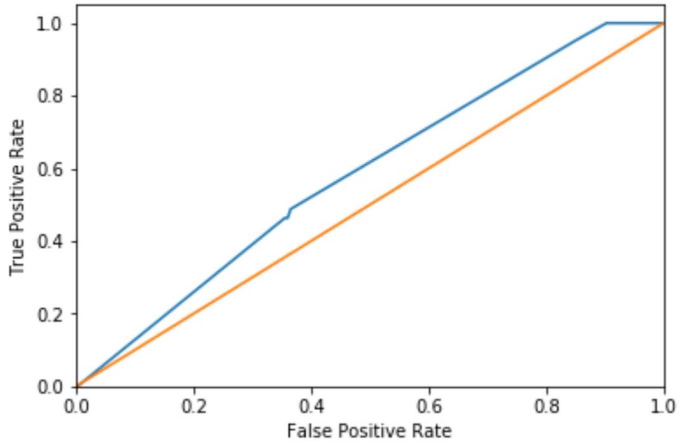

```
Fold: 4

Number of Data points and Variables:
X_train: (4069, 3)
X_test: (452, 3)
y_train: (4069,)
y_test: (452,)

Model: Logistic_Regression
Area under ROC curve: 0.5743993962517295
```



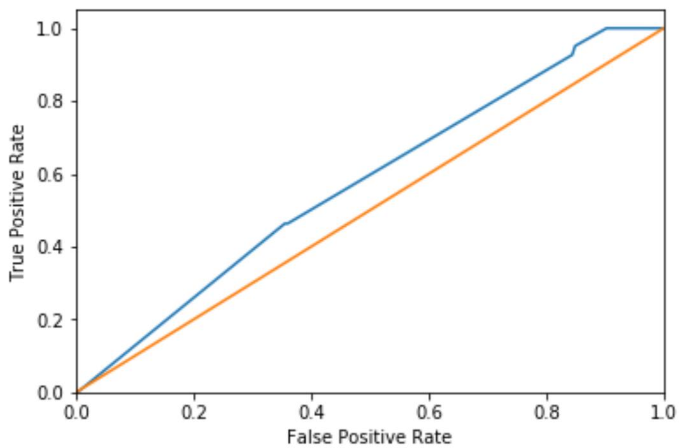

```
Model: Naive_Bayes
Area under ROC curve: 0.5743993962517295
```



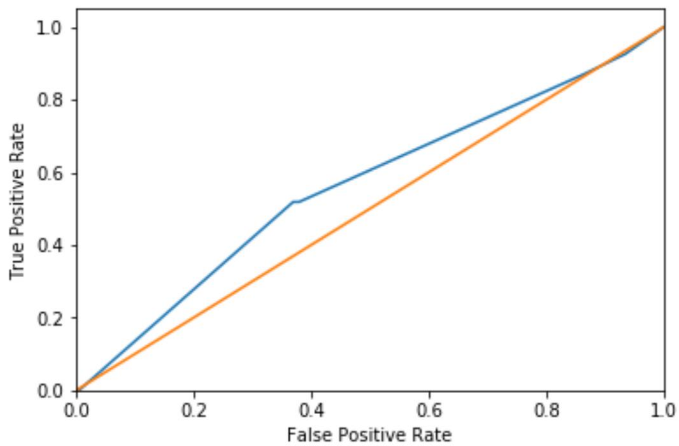

```
Fold: 5

Number of Data points and Variables:
X_train: (4069, 3)
X_test: (452, 3)
y_train: (4069,)
y_test: (452,)

Model: Logistic_Regression
Area under ROC curve: 0.5842383241350662
```



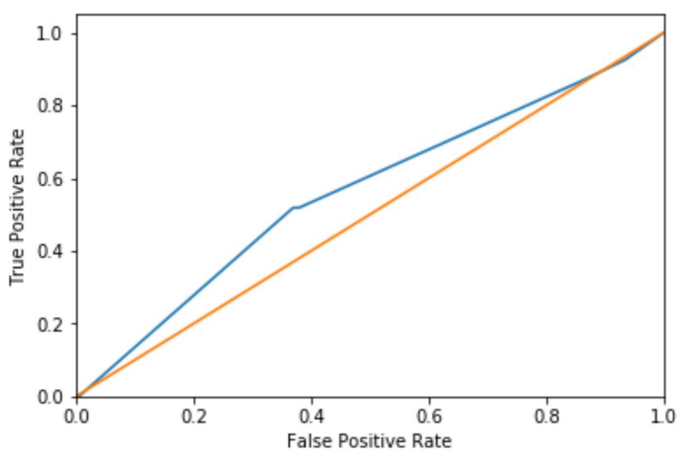

```
Model: Naive_Bayes
Area under ROC curve: 0.5746839950151327
```



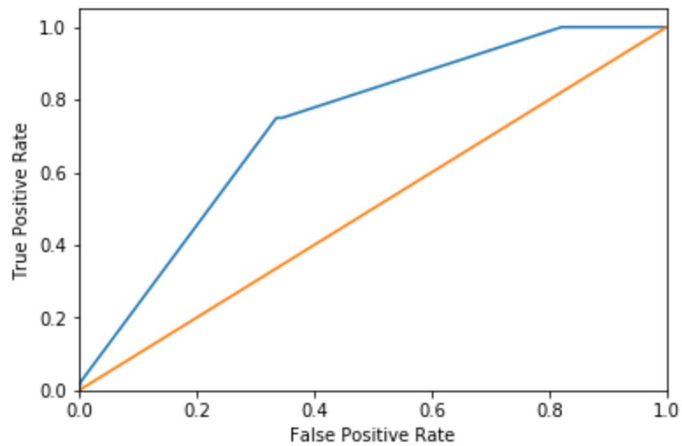

```
Fold: 6

Number of Data points and Variables:
X_train: (4069, 3)
X_test: (452, 3)
y_train: (4069,)
y_test: (452,)

Model: Logistic_Regression
Area under ROC curve: 0.5625116322352502
```



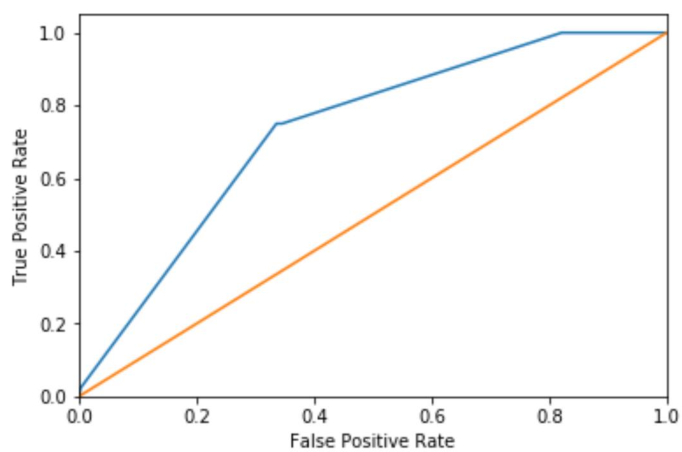

```
Model: Naive_Bayes
Area under ROC curve: 0.5625116322352502
```



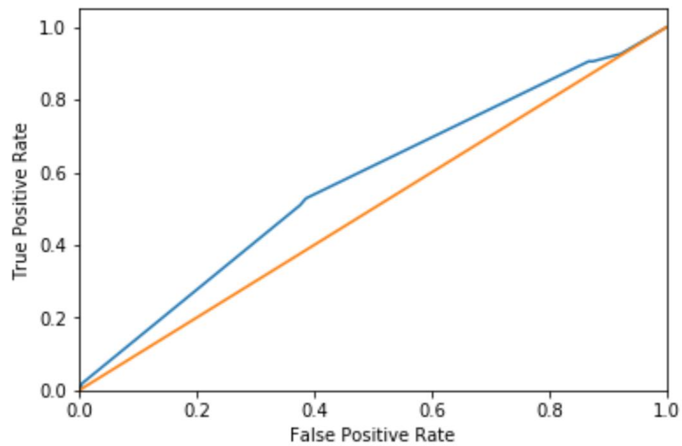

```
Fold: 7

Number of Data points and Variables:
X_train: (4069, 3)
X_test: (452, 3)
y_train: (4069,)
y_test: (452,)

Model: Logistic_Regression
Area under ROC curve: 0.7311441622103387
```



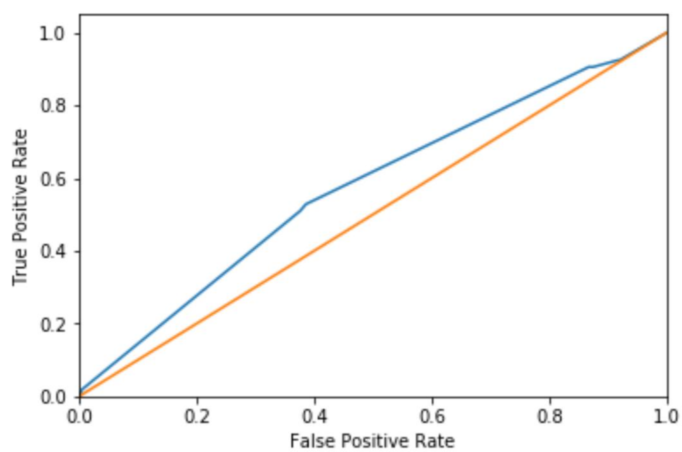

```
Model: Naive_Bayes
Area under ROC curve: 0.7311441622103387
```



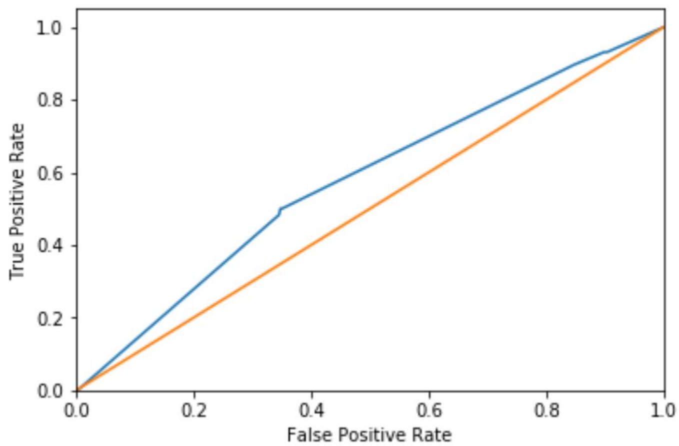

```
Fold: 8

Number of Data points and Variables:
X_train: (4069, 3)
X_test: (452, 3)
y_train: (4069,)
y_test: (452,)

Model: Logistic_Regression
Area under ROC curve: 0.5735328888258382
```



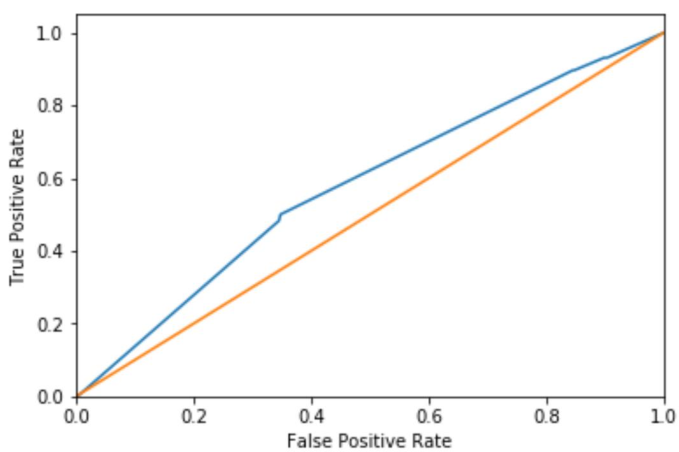

```
Model: Naive_Bayes
Area under ROC curve: 0.5735328888258382
```




```
Fold: 9

Number of Data points and Variables:
X_train: (4069, 3)
X_test: (452, 3)
y_train: (4069,)
y_test: (452,)

Model: Logistic_Regression
Area under ROC curve: 0.5768204095921582
```




```
Model: Naive_Bayes
Area under ROC curve: 0.5778268860493612
```




```
******************************
Average from 10 folds
Method                 Area under ROC Curve
Logistic_Regression    0.611733
Naive_Bayes            0.611060
dtype: float64

Standard Deviation
Logistic_Regression    0.052946
Naive_Bayes            0.053606
dtype: float64
```