Midterm 2

Problem 1

This problem will involve logistic regression on the dataset midterm data 2.csv. The response column is response and all other columns are features.

(a) (5 points) Load the dataset. Remove any unnecessary columns. For any columns that have NA values, fill in the NA values with the median over all non-missing entries in the columns. Format all columns with string entries as categorical variables. Make response a categorical variable. Split the dataset into a training set (75% of observations) and validation set (25% of observations).

In [1]:

```
#%% Step 1 - Load and Clean Data
import pandas as pd;
#temp names to better work with OLS logistic regression ouput
colnames=['row','response', 'a', 'b', 'c', 'd', 'e', 'f', 'g','h', 'i']
file_path = 'C:/Users/danma/Downloads/midterm_data_2.csv'
df = pd.read_table(file_path, sep=",",names=colnames)
df = df.iloc[1: , :]
df = df.loc[:, df.columns != 'row']
del file_path

#get median from columns after dropping NA
nan_values = df[df.isna().any(axis=1)]
print("\nFinding NaN values that have to be replaced in the dataframe")
print(nan_values)
print("\nAfter printing we can see that feat.b and feat.d are only rows with nan value
df_no_NA = df.dropna()
#go back on original and Load NAs with Median Value
values = {"b": df_no_NA["b"].median(), "d": df_no_NA["d"].median()}
df = df.fillna(value=values)
#reassign values
df[['d','response']] = df[['d','response']].astype(int)
df[['a','b','e','f','h','i']] = df[['a','b','e','f','h','i']].astype(float)
del df_no_NA, nan_values, values
#%% Step 2 - Format feat.c and feat.g as Categorical
from sklearn.preprocessing import OneHotEncoder;
oe = OneHotEncoder()
#encode C
encoded_C = oe.fit_transform(df[["c"]])
encoded_C = pd.DataFrame(encoded_C.toarray(),columns=["c_a","c_b","c_c","c_d"])
df = df.join(encoded_C,how='left')
#encode G
encoded_G = oe.fit_transform(df[["g"]])
encoded_G = pd.DataFrame(encoded_G.toarray(),columns=["g_x","g_y","g_z"])
df = df.join(encoded_G,how='left')
#drop original categorical columns
df = df.loc[:, df.columns != "c"]
df = df.loc[:, df.columns != "g"]
#drops na values
df = df.dropna()
print("\nFinal Data Frame (after encoding categorical columns):\n",df.head())
del encoded_C, encoded_G, oe
#%% Step 3 - Split Data
#split into  and Y
x = df.loc[:, df.columns != 'response']
y = df['response']
#turns y into a 1-d array instead of a dataframe column for logistic regression
y = y.to_numpy()
y = y.ravel()
#split into train test split
from sklearn.model_selection import train_test_split as TTS;
TS = 0.25 #for tuning
print("\nTest Size = ", TS, "\n")
train, test = TTS(df, test_size=0.25)
x_train = train.loc[:, df.columns != 'response']
```

```
y_train = train['response']
x_test = test.loc[:, df.columns != 'response']
y_test = test['response']
```

```
Finding NaN values that have to be replaced in the dataframe
    response                  a                  b  c    d  \
12         1    2.07944148117209               NaN  a    1
23         1   -2.07801334492172   -1.9021419904938  a  NaN
45         1    4.62388599491497               NaN  a    1
49         1    3.33989535500895  -1.37236427536683  a  NaN
134        1  -0.267490497018875  -5.47355595390678  c  NaN
203        1   0.204564829910941               NaN  d    0
209        0    5.95272240201007  -4.73429930681862  a  NaN
222        1    2.85395745149959  -5.48939252463402  d  NaN
244        0   -2.41191086199724               NaN  b    0
405        1    3.01208790620179  -3.65857759008987  a  NaN
502        0   -1.98109577329614               NaN  c    0
689        0  0.0360254734948302  -6.01443626601951  c  NaN
700        1   -3.84211836985736               NaN  c    1
785        0   -2.95466218237726  -3.69977968708787  c  NaN
876        0    6.85216301920346               NaN  a    0
901        1   -2.04234251798442  -1.69400869675608  d  NaN
956        1   -3.86557477737264               NaN  d    0

                      e                   f  g                  h  \
12    -1.50726683426877    6.22296083638722  y   7.53599304058386
23     2.86951846250787   -1.92966234079909  z   11.7140118102612
45     1.04513073722725    -7.3956979587732  z   11.5830634963727
49   -0.718159752236264   -7.96202503319852  x   9.40922647043778
134   -1.68092931065423    5.54214592613533  y   12.3845266306818
203    -2.2958515682967    4.34394348007575  z   11.5792135930395
209  -0.968926861374619   -6.48340777540804  y   10.4166989666936
222   0.950981382915278   -3.47949422201344  z    9.3636733380124
244   -3.72387271380726   -8.91333111828326  z   10.3422921083378
405   0.205067494317357    21.5679358255983  x   8.51667113878735
502   -3.31357343321563    -11.587231586493  y   7.62754998087363
689  -0.114272526178723   -12.6105067582284  x   7.37737427188661
700    1.85825936710615   -3.38402624760608  z   8.34189741738764
785    -5.1665747546781   -11.6634815549652  x   9.22189480693221
876    1.25506196793807   -13.1630643685062  x   15.8578014753093
901   -4.12941417059375   0.693296323205981  y   9.95202104786817
956    1.08976706766334   -11.8226771138299  z   10.9994312924781

                      i
12    -1.50912329850252
23     2.84717876827775
45     1.05378518825973
49   -0.666252078684777
134   -1.65741750808627
203   -2.25199207299951
209  -0.973196524673163
222   0.991381439401866
244   -3.75103900095935
405   0.182806232668883
502   -3.35538544415657
689  -0.121070934878067
700    1.81182001703388
785   -5.17404697375275
876    1.23601814159732
901   -4.25173908044909
956       1.07928709624
```

After printing we can see that feat.b and feat.d are only rows with nan values. So we take the median of the dropped NA dataframe and fill those nan with medians.

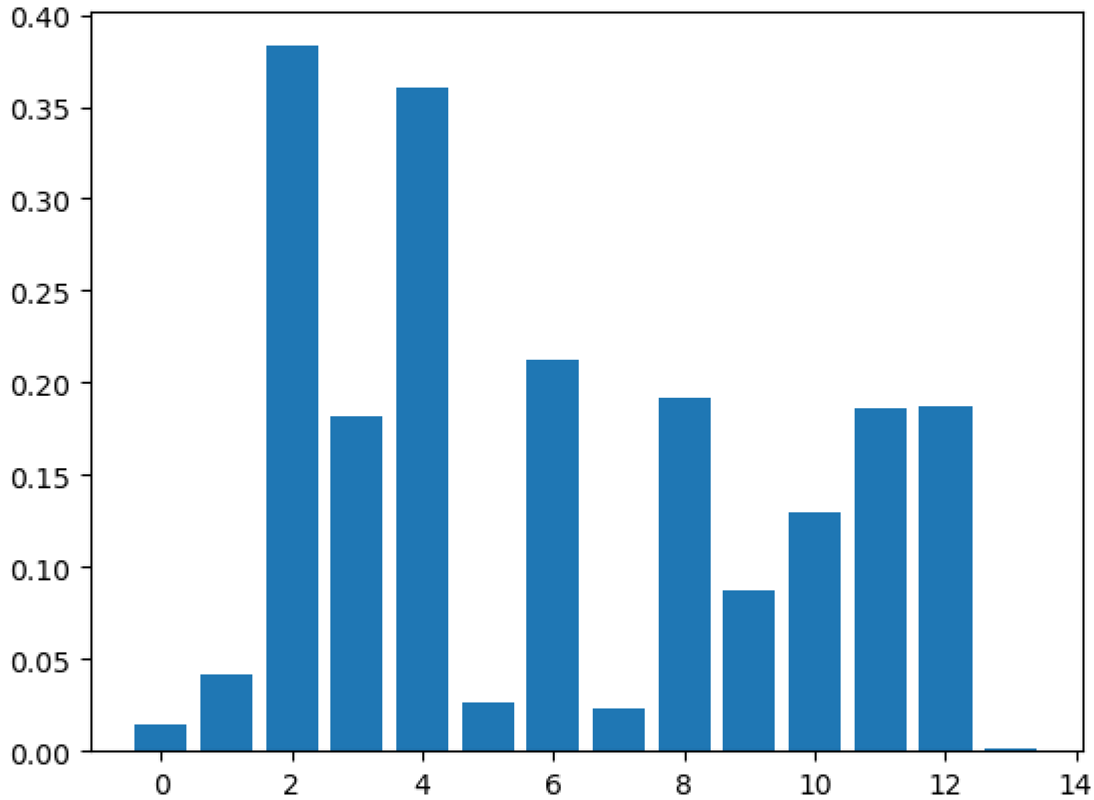
```
Final Data Frame (after encoding categorical columns):
    response         a         b  d         e          f          h         i  \
1          1 -0.681427 -5.493698  0 -0.800615  -4.427602  10.254199 -0.828073
2          1  0.309468 -5.559933  1 -1.155514  -0.799094   9.084749 -1.109698
3          1  5.676125 -4.026970  1 -3.396331  -0.631966   8.753848 -3.417417
4          1  1.211525 -4.198263  1 -1.894569 -16.273262  12.191295 -1.904801
5          1  1.387863 -7.824014  1  4.696980 -22.208877   9.626686  4.715903

   c_a  c_b  c_c  c_d  g_x  g_y  g_z
1  0.0  0.0  0.0  1.0  1.0  0.0  0.0
2  0.0  1.0  0.0  0.0  0.0  1.0  0.0
3  1.0  0.0  0.0  0.0  0.0  1.0  0.0
4  0.0  0.0  1.0  0.0  0.0  0.0  1.0
5  0.0  0.0  1.0  0.0  0.0  1.0  0.0

Test Size =  0.25
```

(b) (5 points) Make a model using all features. Narrow down your features to make a reduced model that uses only the most relevant predictors.

In [2]:

```
#%% Step 4 - Create Full and Reduced Model
print("Complete Logistic Regression Feature Importance:")
from sklearn.linear_model import LogisticRegression;
from matplotlib import pyplot
#using newton-cg to mitigate error with number of samples on default
completemodel = LogisticRegression(max_iter=5000).fit(x_train, y_train)

from sklearn.feature_selection import RFE
rfe = RFE(completemodel, n_features_to_select=7)
rfe.fit(x, y)
print("\nFeature Importance Ranking\n\n")
print(rfe.support_)

# Get Importance
importance = completemodel.coef_[0]

colnames = list(x_train.columns)
# summarize feature importance
for i,v in enumerate(importance):
        print(colnames[i],'Score: %.5f' % (abs((v))))
# plot feature importance
pyplot.bar([x for x in range(len(importance))], abs(importance))
pyplot.show()

del i, v, importance, colnames, rfe
#%% Step 5 - Create Linear Model and Show Summary Screen
import statsmodels.api as sm;

completemodel = sm.Logit(y_train,x_train).fit()
print(completemodel.summary())
```

```
Complete Logistic Regression Feature Importance:

Feature Importance Ranking


[False False  True  True  True False  True  True  True False False False
  True False]
a Score: 0.01475
b Score: 0.04122
d Score: 0.38289
e Score: 0.18198
f Score: 0.36072
h Score: 0.02647
i Score: 0.21246
c_a Score: 0.02297
c_b Score: 0.19148
c_c Score: 0.08699
c_d Score: 0.12980
g_x Score: 0.18612
g_y Score: 0.18753
g_z Score: 0.00093
```




```
Optimization terminated successfully.
         Current function value: 0.364904
         Iterations 7
                           Logit Regression Results
==============================================================================
Dep. Variable:               response   No. Observations:                  749
Model:                          Logit   Df Residuals:                      736
Method:                           MLE   Df Model:                           12
Date:                Wed, 23 Nov 2022   Pseudo R-squ.:                  0.4688
Time:                        20:10:54   Log-Likelihood:                -273.31
converged:                       True   LL-Null:                       -514.56
Covariance Type:            nonrobust   LLR p-value:                 1.176e-95
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
a              0.0139      0.037      0.377      0.706      -0.058       0.086
b             -0.0413      0.072     -0.576      0.565      -0.182       0.099
d              0.4031      0.219      1.843      0.065      -0.026       0.832
e             -3.2314      2.794     -1.156      0.247      -8.708       2.245
f              0.3626      0.027     13.607      0.000       0.310       0.415
h             -0.0247      0.054     -0.454      0.650      -0.131       0.082
i              3.2648      2.797      1.167      0.243      -2.218       8.748
c_a            1.0812   2.17e+07   4.99e-08      1.000   -4.25e+07    4.25e+07
c_b            0.8906   2.17e+07   4.11e-08      1.000   -4.25e+07    4.25e+07
c_c            1.1921   2.17e+07    5.5e-08      1.000   -4.25e+07    4.25e+07
c_d            1.2164   2.17e+07   5.61e-08      1.000   -4.25e+07    4.25e+07
g_x            1.2756   2.17e+07   5.89e-08      1.000   -4.25e+07    4.25e+07
g_y            1.6492   2.17e+07   7.61e-08      1.000   -4.25e+07    4.25e+07
g_z            1.4554   2.17e+07   6.72e-08      1.000   -4.25e+07    4.25e+07
==============================================================================
```

After testing for feature importance, viewing the p-values, and testing with feature selection we see that feat.d and feat.f are the most relevant predictors while feat.e, and feat.i both passed 2 out of 3 selection methods. Therefore our reduce model will contain features d,e,f,i.

In [3]:

```python
#%% Step 6 - Create a Reduced Model
import statsmodels.formula.api as smf

reducedmodel = smf.logit('response ~ d + e + f + i', data=train).fit()
print(reducedmodel.summary())
```

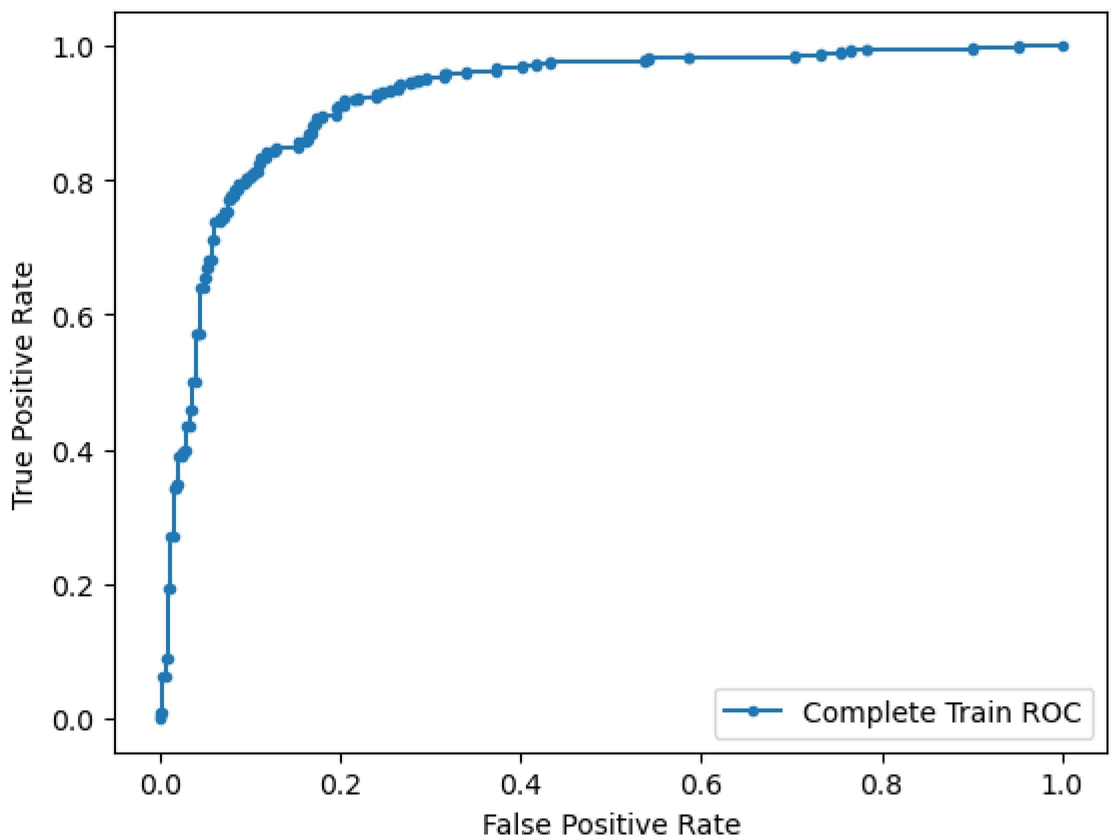
```
Optimization terminated successfully.
         Current function value: 0.367865
         Iterations 7
                           Logit Regression Results
==============================================================================
Dep. Variable:               response   No. Observations:                  749
Model:                          Logit   Df Residuals:                      744
Method:                           MLE   Df Model:                            4
Date:                Wed, 23 Nov 2022   Pseudo R-squ.:                  0.4645
Time:                        20:10:54   Log-Likelihood:                -275.53
converged:                       True   LL-Null:                       -514.56
Covariance Type:            nonrobust   LLR p-value:                3.729e-102
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      2.4556      0.234     10.498      0.000       1.997       2.914
d              0.4141      0.215      1.923      0.055      -0.008       0.836
e             -3.4361      2.757     -1.246      0.213      -8.840       1.968
f              0.3596      0.026     13.698      0.000       0.308       0.411
i              3.4795      2.760      1.261      0.207      -1.931       8.890
==============================================================================
```

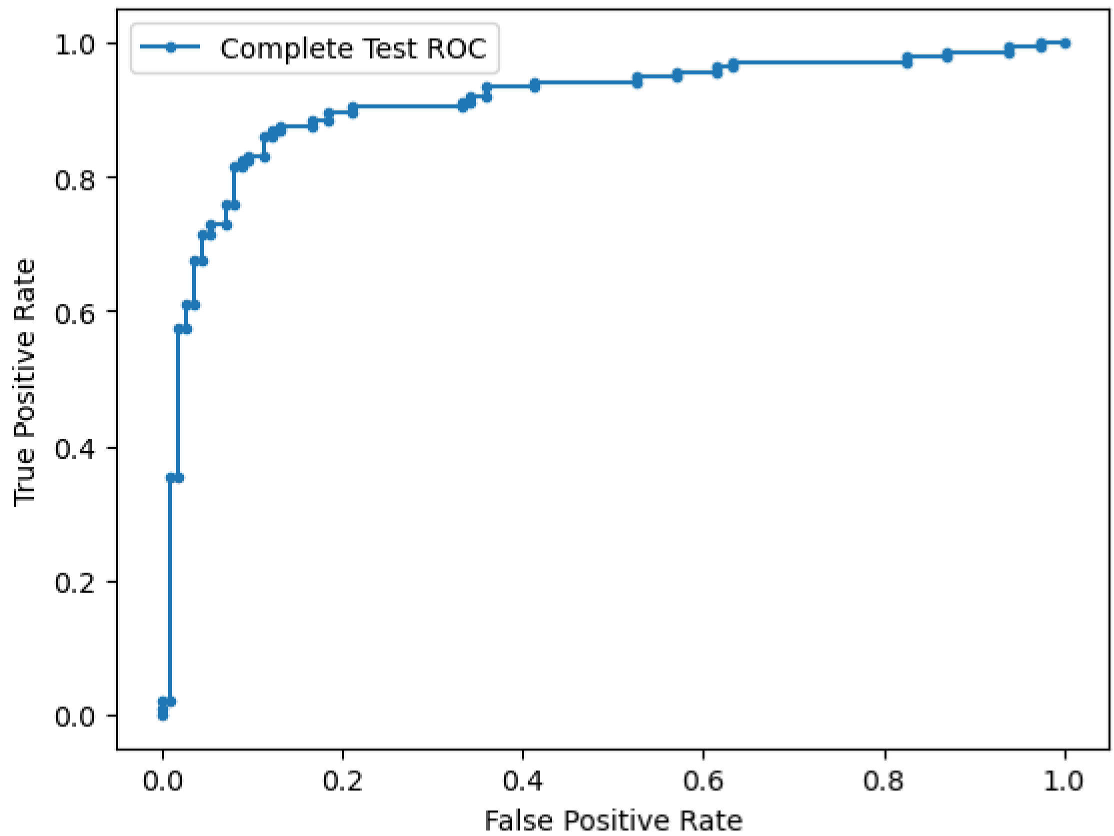
(c) (5 points) Create an ROC curve for your full and reduced model on both the training and validation sets (4 curves in all). Comment on the degree of overfitting for validation performance vs. training performance and the adequacy of your reduced model compared to your full model.

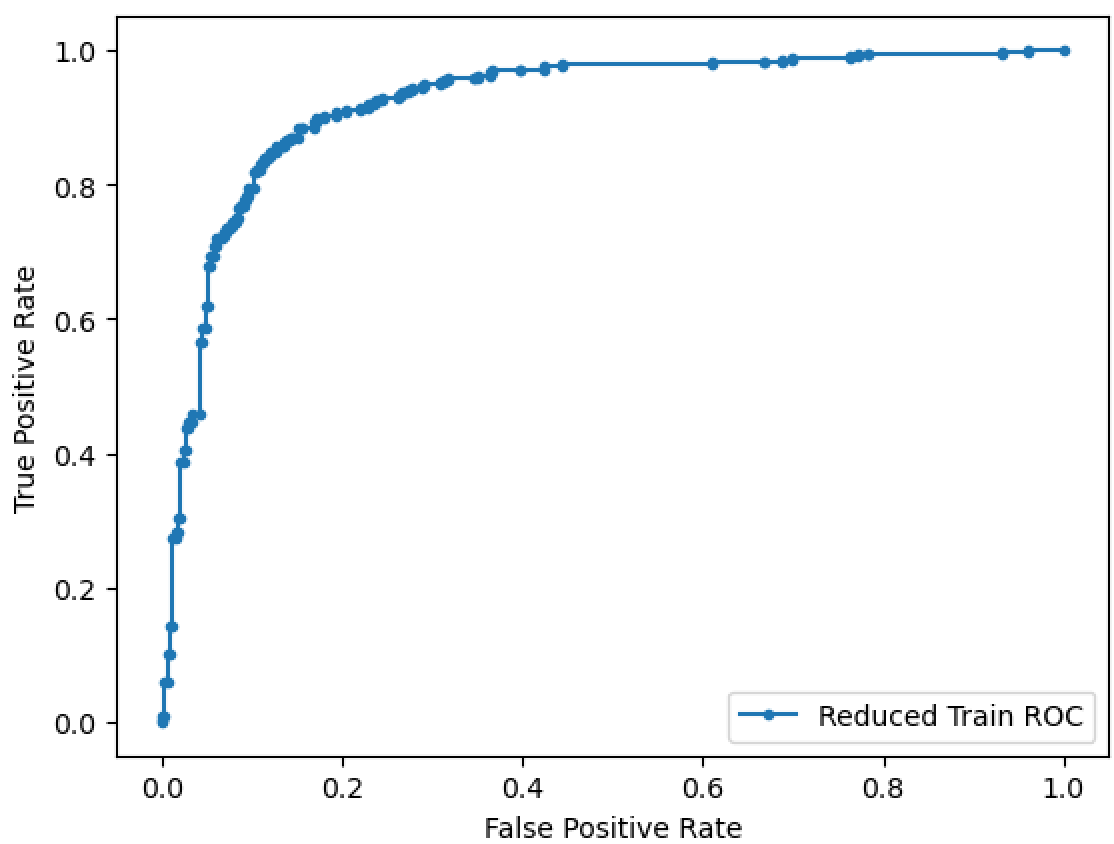
In [4]:

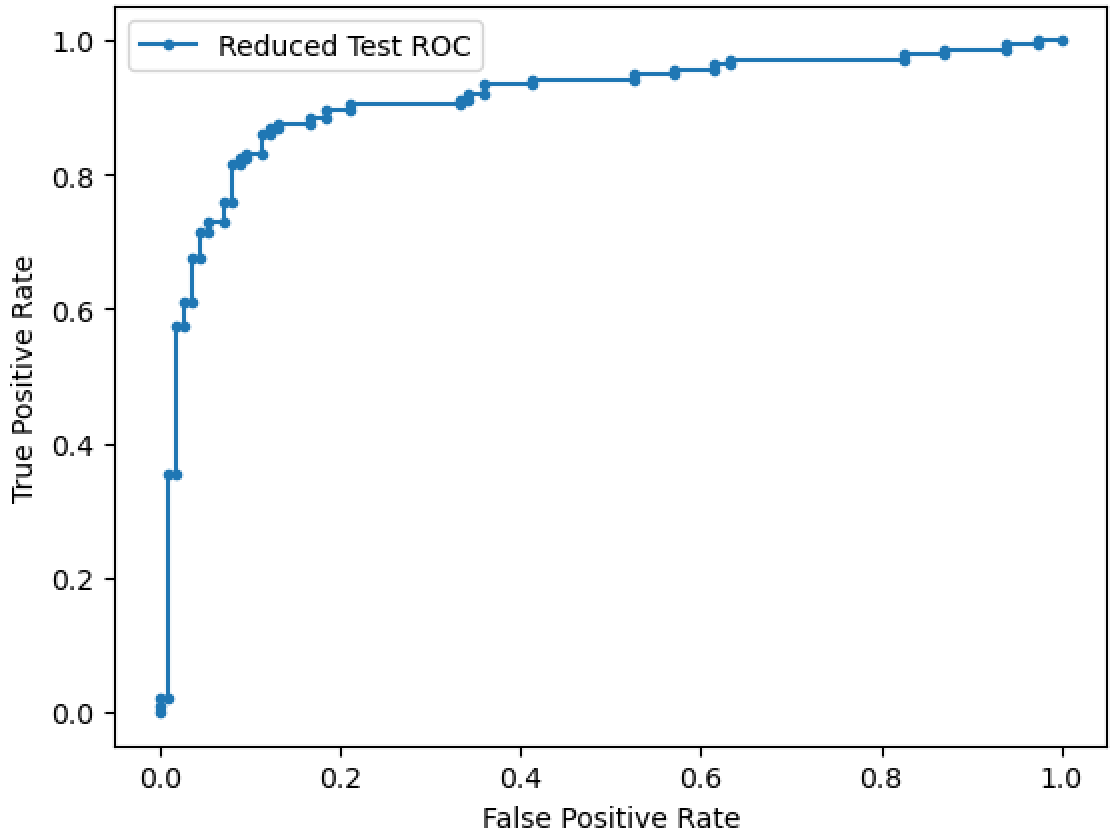
```
#%% Step 7 - Create ROC Curve
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

#Complete Model Training
comp_train_pred = completemodel.predict(x_train)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_train.astype('int32'), comp_train_pred)
pyplot.plot(fpr, tpr, marker='.', label='Complete Train ROC')
# axis labels
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')
# show the legend
pyplot.legend()
# show the plot
pyplot.show()

#to create summary table at the end
from tabulate import tabulate

data = {'Complete Train': [roc_auc_score(y_train.astype('int32'), comp_train_pred)]}
table = pd.DataFrame(data)

#Complete Model Testing
comp_test_pred = completemodel.predict(x_test)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test.astype('int32'), comp_test_pred)
pyplot.plot(fpr, tpr, marker='.', label='Complete Test ROC')
# axis labels
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')
# show the legend
pyplot.legend()
# show the plot
pyplot.show()

data = {'Complete Test': [roc_auc_score(y_test.astype('int32'), comp_test_pred)]}
table['Complete Test'] = pd.DataFrame(data)

#Reduced Model Training
red_train_pred = reducedmodel.predict(x_train)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_train.astype('int32'), red_train_pred)
pyplot.plot(fpr, tpr, marker='.', label='Reduced Train ROC')
# axis labels
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')
# show the legend
pyplot.legend()
# show the plot
```

```
pyplot.show()

data = {'Reduced Train': [roc_auc_score(y_train.astype('int32'), red_train_pred)]}
table['Reduced Train'] = pd.DataFrame(data)

#Reduced Model Testing
red_test_pred = reducedmodel.predict(x_test)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test.astype('int32'), comp_test_pred)
pyplot.plot(fpr, tpr, marker='.', label='Reduced Test ROC')
# axis labels
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')
# show the legend
pyplot.legend()
# show the plot
pyplot.show()

data = {'Reduced Test': [roc_auc_score(y_test.astype('int32'), red_test_pred)]}
table['Reduced Test'] = pd.DataFrame(data)

print("\n\nAUC Score Results:\n")

print(tabulate(table, headers='keys',tablefmt='fancy_grid',showindex=["AUC"]))

del comp_test_pred, comp_train_pred, fpr, red_test_pred, red_train_pred, thresholds, t
```




Complete Test ROC
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0


Reduced Train ROC
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0




```
AUC Score Results:
```

	Complete Train	Complete Test	Reduced Train	Reduced Test
AUC	0.92325	0.911765	0.922102	0.914796

Both models appear to have similar degrees of fitting, where the gaps between the AUC's are .01. This gap is found between the validation/test and training AUC's from their ROC curve. Having said that, neither model shows a degree of overfitting due to having a small gap between AUC's. Seeing as the results were very simliar both complete and reduced appear adequate, but one would choose the reduced as it is the more parsimonious.

(d) (5 points) Using your reduced model, perform predictions for P(response = 1|features) for the validation set. Perform predictions for the binary response by thresholding your predicted probabilities P(response = 1|features) at two different values: 0.5 and 0.65. Calculate the overall prediction accuracy for both thresholds. Calculate the False Negative Rate for both thresholds.

In [5]:

```
#%% Step 8 - Threshold Comparison
from sklearn.metrics import accuracy_score
# Predicted probability
y_predict_prob = reducedmodel.predict(x_test)
print("Define threshold 0.5")
y_predict_class = [1 if prob > 0.5 else 0 for prob in y_predict_prob]
print("Accuracy:", round(accuracy_score(y_test, y_predict_class), 3))

from sklearn.metrics import confusion_matrix
CM = confusion_matrix(y_test, y_predict_class)
FN = CM[1][0]
print("False Negative Rate",FN)

print("\nDefine threshold 0.65")
y_predict_class = [1 if prob > 0.65 else 0 for prob in y_predict_prob]
print("Accuracy:", round(accuracy_score(y_test, y_predict_class), 3))
CM = confusion_matrix(y_test, y_predict_class)
```

```
FN = CM[1][0]
print("False Negative Rate",FN)
```

```
Define threshold 0.5
Accuracy: 0.864
False Negative Rate 20

Define threshold 0.65
Accuracy: 0.832
False Negative Rate 35
```

(e) (5 points) Make two altered copies of your validation set: one where feat.d is set to 1 for all rows, and another where feat.d is set to 0 for all rows. All other columns should remain the same as your original validation set. Using your reduced model, perform predictions for P(response = 1|features) for both altered validation sets, and average the predicted probabilities across all validation observations (end up with 2 average probabilities, one for each altered dataset). Finally, calculate the difference between these average probabilities (either order for the subtraction is OK). How can you interpret the average difference that you have found?

In [6]:

```
#%% Step 9 - feat.d Manipulation

print("feat.d set to 0 results:")
d_set_0 = x_test.assign(d=0)
y_predict_prob0 = reducedmodel.predict(d_set_0)
print("Prediction Probability Average:",round(y_predict_prob0.mean(),4))

print("\nfeat.d set to 1 results:")
d_set_1 = x_test.assign(d=1)
y_predict_prob1 = reducedmodel.predict(d_set_1)
print("Prediction Probability Average:",round(y_predict_prob1.mean(),4))

print("\nDifference Between Probabilities ",round((y_predict_prob1.mean() - y_predict_
```

```
feat.d set to 0 results:
Prediction Probability Average: 0.4952

feat.d set to 1 results:
Prediction Probability Average: 0.5437

Difference Between Probabilities  0.0484
```

We can interpret this small average difference by saying that feat.d has a similar probability in being 1 as it is 0, therefore showing the feat.d is almost evenly distributed when viewing predictions based off of the column. Due to it having a higher probability for 1 we can assume that there are more 1's for feat.d for accuracys sake.