In [1]:
```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_squared_error

from sklearn.metrics import accuracy_score, classification_report
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import mutual_info_regression

import os
os.getcwd()
```

Out[1]:
```
'C:\\Users\\csant\\python_analysis_update'
```

In [2]:
```
# Load CSV file of curated/merged data
df = pd.read_csv('c:/csc606/curated colocated data/merged_velocity_data.csv',low_me
```

In [3]:
```
# Create single column in dataframe called "key" that concatenates leg, site, hole,
# The key column will be used for joining/merging later
df['key']=df['leg'] + "." + df['site'] + "." + df['hole'] + "." + df['core'] + "."
```

In [4]:
```
# Create the rho feature, as recommended by Taylor
# rho = absolute difference between GRA density and 0.31 * velocity ^ 0.25
df['rho']=abs(df['GRA_density(g/cm^3)']-(0.31*df['compressional_velocity(m/s)']**0.
```

In [5]:
```
df
```

Out[5]:

	latitude(dd)	longitude(dd)	leg	site	hole	core	section	depth_m	compr
0	31.789787	139.026217	350	U1437	D	24	4	645.330	
1	31.789787	139.026217	350	U1437	D	53	1	912.683	
2	31.789798	139.026523	350	U1437	E	5	3	1113.300	
3	-38.829700	178.476055	372A	U1517	C	8	2	41.092	
4	-38.859490	178.896032	375	U1518	F	30	7	472.850	
...	...	...	...	...	...	...	...	...	
26157261	19.489617	-82.936100	165	998	A	7	5	62.610	
26157262	16.553717	-79.867400	165	1000	A	9	3	73.100	
26157263	55.477183	-14.650867	162	981	A	7	1	55.150	
26157264	1.202300	-83.737000	111	677	A	27	5	247.550	
26157265	16.130700	60.744000	117	720	A	1	4	4.900	

26157266 rows × 21 columns

In [6]:

```
# Load CSV file of Taylor's labeled data
labeled_df = pd.read_csv('c:/csc606/image_assessment_augmented.csv',low_memory=Fals

# convert int columns to str
labeled_df['leg']=labeled_df['leg'].astype(str)
labeled_df['site']=labeled_df['site'].astype(str)
labeled_df['hole']=labeled_df['hole'].astype(str)
labeled_df['core']=labeled_df['core'].astype(str)
labeled_df['section']=labeled_df['section'].astype(str)

# Create single column in dataframe called "key" that concatenates leg, site, hole,
# The key column will be used for joining/merging later
labeled_df['key']=labeled_df['leg'] + "." + labeled_df['site'] + "." + labeled_df['
```

In [7]:

```
labeled_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 131 entries, 0 to 130
Data columns (total 9 columns):
 #   Column                       Non-Null Count  Dtype 
---  ------                       --------------  ----- 
 0   leg                          131 non-null    object
 1   site                         131 non-null    object
 2   hole                         131 non-null    object
 3   core                         131 non-null    object
 4   section                      131 non-null    object
 5   file_name                    131 non-null    object
 6   greater_than_50_percent_bad  131 non-null    bool  
 7   notes                        58 non-null     object
 8   key                          131 non-null    object
dtypes: bool(1), object(8)
memory usage: 8.4+ KB
```

In [8]:

```
labeled_df
```

Out[8]:

	leg	site	hole	core	section	file_name	greater_th
0	166	1006	A	13	4	leg166_site1006_holeA_core13_section4.png	
1	167	1010	B	1	3	leg167_site1010_holeB_core1_section3.png	
2	167	1010	C	4	3	leg167_site1010_holeC_core4_section3.png	
3	167	1010	E	10	4	leg167_site1010_holeE_core10_section4.png	
4	167	1010	E	9	3	leg167_site1010_holeE_core9_section3.png	
...	...	...	...	...	...	...	
126	398	U1591	B	5	4	leg398_siteU1591_holeB_core5_section4.png	
127	398	U1593	A	34	1	leg398_siteU1593_holeA_core34_section1.png	
128	398	U1598	A	8	1	leg398_siteU1598_holeA_core8_section1.png	
129	398	U1600	B	19	1	leg398_siteU1600_holeB_core19_section1.png	
130	398	U1600	B	5	1	leg398_siteU1600_holeB_core5_section1.png	

131 rows × 9 columns

In [9]:

```
# Here we merge the labeled dataset with the curated dataset
merged_labeled_df = pd.merge(labeled_df, df, on='key', how='left')
```

In [10]:

```
merged_labeled_df
```

Out[10]:

	leg_x	site_x	hole_x	core_x	section_x	file_name	gr
0	166	1006	A	13	4	leg166_site1006_holeA_core13_section4.png	
1	166	1006	A	13	4	leg166_site1006_holeA_core13_section4.png	
2	166	1006	A	13	4	leg166_site1006_holeA_core13_section4.png	
3	166	1006	A	13	4	leg166_site1006_holeA_core13_section4.png	
4	166	1006	A	13	4	leg166_site1006_holeA_core13_section4.png	
...	...	...	...	...	...	...	
7015	398	U1600	B	5	1	leg398_siteU1600_holeB_core5_section1.png	
7016	398	U1600	B	5	1	leg398_siteU1600_holeB_core5_section1.png	
7017	398	U1600	B	5	1	leg398_siteU1600_holeB_core5_section1.png	
7018	398	U1600	B	5	1	leg398_siteU1600_holeB_core5_section1.png	
7019	398	U1600	B	5	1	leg398_siteU1600_holeB_core5_section1.png	

7020 rows × 29 columns

In [11]:

```
# here we create the features:  mean/mode/median/std/min/max for both depth and com
# we also create the 25%(Q1) and 75%(Q3) quantiles for compressional velocity withi
#
# added in v2:  mean/mode/median/std/min/max for the calculated rho feature
#
groupby_columns = ['key','greater_than_50_percent_bad']
df_grouped = merged_labeled_df.groupby(groupby_columns)[['depth_m','compressional_v
    depth_mean=('depth_m','mean'),
    depth_median=('depth_m','median'),
    depth_mode=('depth_m',lambda x: x.mode()[0]),
    depth_std=('depth_m', 'std'),
    depth_min=('depth_m', 'min'),
    depth_max=('depth_m', 'max'),
    velocity_mean=('compressional_velocity(m/s)','mean'),
    velocity_median=('compressional_velocity(m/s)','median'),
    velocity_mode=('compressional_velocity(m/s)',lambda x: x.mode()[0]),
    velocity_std=('compressional_velocity(m/s)', 'std'),
    velocity_min=('compressional_velocity(m/s)', 'min'),
    velocity_max=('compressional_velocity(m/s)', 'max'),
    velocity_q1 =('compressional_velocity(m/s)',lambda x: x.quantile(0.25)),
    velocity_q3 =('compressional_velocity(m/s)',lambda x: x.quantile(0.75)),
    rho_mean=('rho','mean'),
    rho_median=('rho','median'),
    rho_mode=('rho',lambda x: x.mode()[0]),
    rho_std=('rho','std'),
```

```
    rho_min=('rho','min'),
    rho_max=('rho','max')
)
```

In [12]:
```
df_grouped
```

Out[12]:

key	greater_than_50_percent_bad	depth_mean	depth_median	depth_mode
166.1006.A.13.4	**False**	116.851433	116.8715	116.150
167.1010.B.1.3	**False**	3.480000	3.4800	3.030
167.1010.C.4.3	**False**	28.210000	28.2100	27.530
167.1010.E.10.4	**True**	90.240000	90.2400	89.550
167.1010.E.9.3	**False**	79.240000	79.2400	78.550
...	**...**	...	...	...
398.U1591.B.5.4	**False**	40.932646	40.9330	40.354
398.U1593.A.34.1	**True**	227.806071	227.8125	227.100
398.U1598.A.8.1	**False**	61.359717	61.3500	60.700
398.U1600.B.19.1	**False**	87.416433	87.4240	86.700
398.U1600.B.5.1	**True**	21.648475	21.6480	20.900

131 rows × 20 columns

In [13]:
```
# calculate the interquartile range for compressional velocity between Q1 and Q3
# calculate upper and lower boundaries for compressional velocity which are 1.5x th
#   and 1.5x the IGR below the median
df_grouped['velocity_igr'] = df_grouped['velocity_q3']-df_grouped['velocity_q1']
df_grouped['velocity_upper'] = df_grouped['velocity_median'] + (df_grouped['velocit
df_grouped['velocity_lower'] = df_grouped['velocity_median'] - (df_grouped['velocit
```

In [14]:
```
df_grouped
```

Out[14]:

key	greater_than_50_percent_bad	depth_mean	depth_median	depth_mode
166.1006.A.13.4	False	116.851433	116.8715	116.150
167.1010.B.1.3	False	3.480000	3.4800	3.030
167.1010.C.4.3	False	28.210000	28.2100	27.530
167.1010.E.10.4	True	90.240000	90.2400	89.550
167.1010.E.9.3	False	79.240000	79.2400	78.550
...	...	...	...	...
398.U1591.B.5.4	False	40.932646	40.9330	40.354
398.U1593.A.34.1	True	227.806071	227.8125	227.100
398.U1598.A.8.1	False	61.359717	61.3500	60.700
398.U1600.B.19.1	False	87.416433	87.4240	86.700
398.U1600.B.5.1	True	21.648475	21.6480	20.900

131 rows × 23 columns

In [15]:

```
# get rid of the indexes created by groupby
df_grouped = df_grouped.reset_index()

# convert the boolean label to numeric 1 and 0
df_grouped['label'] = df_grouped['greater_than_50_percent_bad'].astype(int)
df_grouped
```

Out[15]:

	key	greater_than_50_percent_bad	depth_mean	depth_median	depth_mod
0	166.1006.A.13.4	False	116.851433	116.8715	116.1
1	167.1010.B.1.3	False	3.480000	3.4800	3.0
2	167.1010.C.4.3	False	28.210000	28.2100	27.5
3	167.1010.E.10.4	True	90.240000	90.2400	89.5
4	167.1010.E.9.3	False	79.240000	79.2400	78.5
...	...	...	...	...	
126	398.U1591.B.5.4	False	40.932646	40.9330	40.3
127	398.U1593.A.34.1	True	227.806071	227.8125	227.1
128	398.U1598.A.8.1	False	61.359717	61.3500	60.7
129	398.U1600.B.19.1	False	87.416433	87.4240	86.7
130	398.U1600.B.5.1	True	21.648475	21.6480	20.9

131 rows × 26 columns

In [16]:
```
# pull out the key identifiers to X_identifiers, since they are text and can't be a
# drop the label columns and the key
X_identifiers = df_grouped['key']
X = df_grouped.drop('greater_than_50_percent_bad', axis=1).drop('key', axis=1).drop

# create the labeled series from the label column
y = df_grouped['label']
```

In [17]:
```
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [18]:
```
# Model selection of Logistic Regression
model = LogisticRegression(max_iter=50000)
```

In [19]:
```
# Model Training
model.fit(X_train, y_train)
```

Out[19]:
```
LogisticRegression
LogisticRegression(max_iter=50000)
```

In [20]:
```
# Model prediction using the test dataset
y_pred = model.predict(X_test)
```

In [21]:
```
# Calculate prediction probabilities from test dataset
y_pred_prob = model.predict_proba(X_test)[:, 1]
```

In [22]:
```
y_pred_prob
```

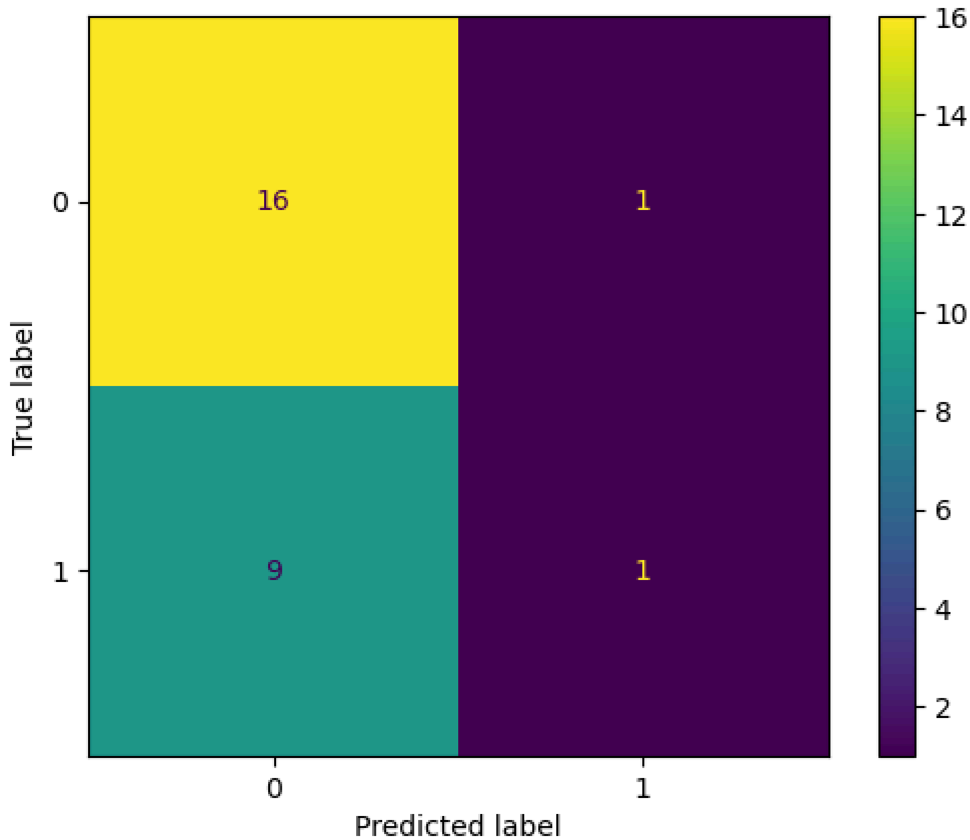
Out[22]:
```
array([2.46717416e-04, 9.94993379e-01, 3.58020699e-01, 1.25775943e-02,
       3.42142605e-02, 4.81295229e-02, 1.13957797e-01, 7.96371636e-02,
       6.17198489e-02, 1.21676697e-01, 1.19821646e-01, 9.74089621e-02,
       5.86016347e-01, 4.24463806e-03, 1.62010579e-01, 1.03997401e-01,
       4.41887674e-02, 2.55587338e-02, 1.74499097e-01, 1.42091907e-01,
       1.92719303e-01, 8.82836554e-02, 9.49209263e-20, 4.17216757e-01,
       3.72961391e-01, 3.38447666e-02, 9.27158845e-02])
```

In [23]:
```
# Calculate accuracy and create a classification report having precision, recall, f
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
```

In [24]:
```
print(f"Accuracy: {accuracy:.2f}")
print("\nClassification Report:\n", report)
```

```
Accuracy: 0.63

Classification Report:
               precision    recall  f1-score   support

           0       0.64      0.94      0.76        17
           1       0.50      0.10      0.17        10

    accuracy                           0.63        27
   macro avg       0.57      0.52      0.46        27
weighted avg       0.59      0.63      0.54        27
```

In [25]:
```
# build confusion matrix for the test dataset
conf_mat = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=conf_mat,
                              display_labels=model.classes_)
disp.plot()
plt.show()
```



In [26]:
```
fs = SelectKBest(score_func=mutual_info_regression, k=4) # k=4 based on results of
fs.fit(X_train, y_train)
selected_features_mask = fs.get_support()
print(X_train.columns[selected_features_mask])
```

```
Index(['velocity_mean', 'velocity_median', 'velocity_q3', 'velocity_upper'], dtype
='object')
```

In [27]:
```
x_train_fs = fs.transform(X_train)
x_test_fs = fs.transform(X_test)
```

In [28]:
```
x_train_fs.shape
```

Out[28]:
```
(104, 4)
```

In [29]:
```
model2 = LogisticRegression(max_iter=50000)
model2.fit(x_train_fs, y_train)
model2.score(x_test_fs, y_test)
```

Out[29]:
```
0.7037037037037037
```

In [30]:
```
model2.score(x_train_fs, y_train)
```

Out[30]:
```
0.9038461538461539
```

In [31]:
```
# Plot the importance of each feature
# Code from:  Murach's Python for Data Science, 2nd Edition, Chapter 11
df1 = pd.DataFrame(X_train.columns[selected_features_mask], columns=['feature'])
```

```python
df2 = pd.DataFrame(fs.scores_[selected_features_mask], columns=['importance'])
importance = df1.join(df2)
importance.sort_values('importance', ascending=False).head()
```

Out[31]:

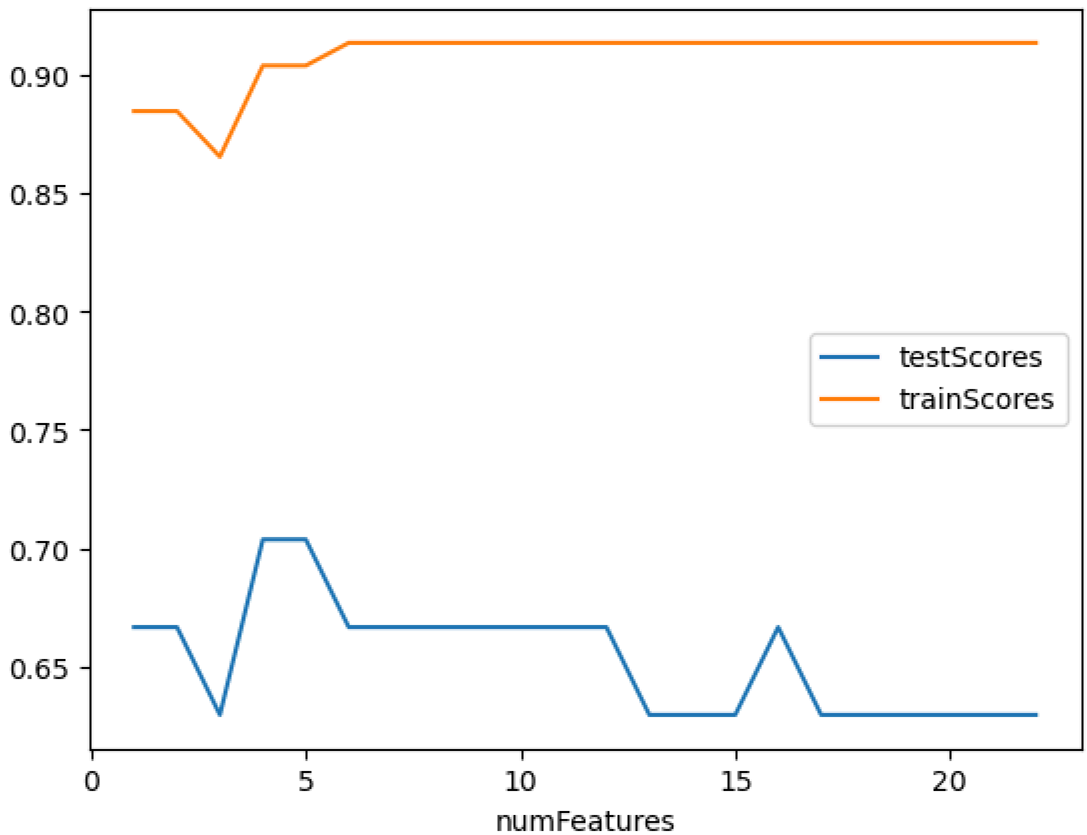
	feature	importance
2	velocity_q3	0.295706
1	velocity_median	0.239710
3	velocity_upper	0.239058
0	velocity_mean	0.232435

In [32]:

```python
# Score the model for test/train datasets for varying number of features
#
# Code from:  Murach's Python for Data Science, 2nd Edition, Chapter 11
#

model2 = LogisticRegression()
testScores = []
trainScores = []

for i in range(1, len(X_train.columns)):
    fs = SelectKBest(score_func=mutual_info_regression, k=i)
    fs.fit(X_train, y_train)

    x_train_fs = fs.transform(X_train)
    x_test_fs = fs.transform(X_test)

    model.fit(x_train_fs, y_train)

    testScore = model.score(x_test_fs, y_test)
    trainScore = model.score(x_train_fs, y_train)
    testScores.append(testScore)
    trainScores.append(trainScore)

df = pd.DataFrame(data={'testScores':testScores, 'trainScores':trainScores})
df.reset_index(inplace=True)
df.rename(columns={'index':'numFeatures'}, inplace=True)
df.numFeatures = df.numFeatures + 1
df.plot(x='numFeatures', y=['testScores','trainScores'])
```

Out[32]:

```
<Axes: xlabel='numFeatures'>
```


0.90
0.85
0.80
0.75
0.70
0.65
0
5
10
15
20
numFeatures
testScores
trainScores

In [ ]:

In [ ]: