CO544-Project

May 10, 2020

[8]:
```
import pandas as pd
import matplotlib.pyplot as plt

#read the dataset
dataset = pd.read_csv('E:/University Works/3rd Year/Semester 6/CO 544 - Machine␣
↪Learning and Data Mining/Project/data.csv',sep= ',')
dataset.head()
```

[8]:

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16
0	b	30.83	u	g	0.00	w	0	True	v	1.25	True	1	False	202	g	Success
1	a	58.67	u	g	4.46	q	560	True	h	3.04	True	6	False	43	g	Success
2	a	24.5	u	g	0.50	q	824	False	h	1.50	True	0	False	280	g	Success
3	b	27.83	u	g	1.54	w	3	True	v	3.75	True	5	True	100	g	Success
4	b	25	u	g	11.25	c	1208	True	v	2.50	True	17	False	200	g	Success

[9]:
```
print ("Dataset Length: ", len(dataset))
print ("Dataset Shape: ", dataset.shape)
```

```
Dataset Length:  552
Dataset Shape:  (552, 16)
```

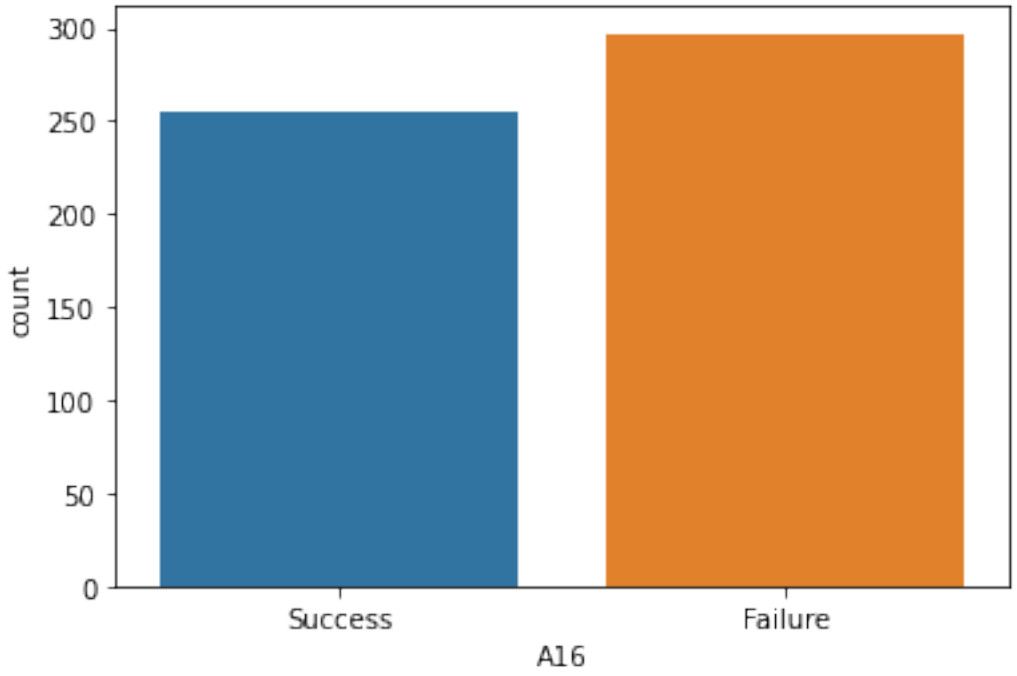
[10]:
```
print(dataset['A16'].unique())
```

```
['Success' 'Failure']
```

[11]:
```
print(dataset.groupby('A16').size())
```

```
A16
Failure    297
Success    255
dtype: int64
```

[12]:
```
import seaborn as sns
sns.countplot(dataset['A16'],label="Count")
plt.show()
```



[13]:
```
dataset.__eq__('?').sum()
```

```
c:\users\user\appdata\local\programs\python\python38\lib\site-
packages\pandas\core\ops\array_ops.py:253: FutureWarning: elementwise comparison
failed; returning scalar instead, but in the future will perform elementwise
comparison
  res_values = method(rvalues)
```

[13]:
```
A1      8
A2     10
A3      4
A4      4
A5      0
A6      6
A7      0
A8      0
A9      6
A10     0
A11     0
A12     0
A13     0
```

```
A14    10
A15     0
A16     0
dtype: int64
```

[14]:
```
dataset.dtypes
```

[14]:
```
A1      object
A2      object
A3      object
A4      object
A5     float64
A6      object
A7       int64
A8        bool
A9      object
A10    float64
A11       bool
A12      int64
A13       bool
A14     object
A15     object
A16     object
dtype: object
```

[15]:
```
#Since A2,A5,A7,A10,A12,A14 has to be numeric
#replace all the missing data with 0

#import numpy as np
dataset[['A2','A5','A7','A10','A12','A14']] =␣
 ↪dataset[['A2','A5','A7','A10','A12','A14']].replace('?',0)
#dataset['A1'].replace('?',np.nan,inplace=True)
#dataset['A2'].replace('?',np.nan,inplace=True)#numeric
#dataset['A3'].replace('?',np.nan,inplace=True)
#dataset['A4'].replace('?',np.nan,inplace=True)
#dataset['A6'].replace('?',np.nan,inplace=True)
#dataset['A9'].replace('?',np.nan,inplace=True)#numeric
#dataset['A14'].replace('?',np.nan,inplace=True)
#dataset.dropna(inplace=True)

#Handle missing values using imputation
#from sklearn.impute import SimpleImputer
#my_imputer = SimpleImputer()
#data_with_imputed_values = my_imputer.fit_transform(original_data)
```

[16]:
```
#change A2,A5,A7,A10,A12,A14 data type to float
dataset['A14'] = dataset.A14.astype(float)
```

```
dataset['A2'] = dataset.A2.astype(float)
dataset['A5'] = dataset.A5.astype(float)
dataset['A7'] = dataset.A7.astype(float)
dataset['A10'] = dataset.A10.astype(float)
dataset['A12'] = dataset.A12.astype(float)
```

[17]:
```
dataset.dtypes
```

[17]:
```
A1      object
A2     float64
A3      object
A4      object
A5     float64
A6      object
A7     float64
A8        bool
A9      object
A10    float64
A11       bool
A12    float64
A13       bool
A14    float64
A15     object
A16     object
dtype: object
```

[18]:
```
# summary statistics of character column
print (dataset.describe(include='all'))
```

```
          A1           A2   A3   A4           A5   A6             A7     A8   A9  \
count    552  552.000000  552  552  552.000000  552     552.000000    552  552
unique     3         NaN    4    4         NaN   15            NaN      2   10
top        b         NaN    u    g         NaN    c            NaN  False    v
freq     379         NaN  416  416         NaN  104            NaN    306  310
mean     NaN   31.397373  NaN  NaN    4.884384  NaN    1100.827899    NaN  NaN
std      NaN   12.831325  NaN  NaN    5.086809  NaN    5628.306468    NaN  NaN
min      NaN    0.000000  NaN  NaN    0.000000  NaN       0.000000    NaN  NaN
25%      NaN   22.580000  NaN  NaN    1.083750  NaN       0.000000    NaN  NaN
50%      NaN   28.210000  NaN  NaN    2.750000  NaN       5.000000    NaN  NaN
75%      NaN   38.960000  NaN  NaN    7.551250  NaN     456.500000    NaN  NaN
max      NaN   80.250000  NaN  NaN   28.000000  NaN  100000.000000    NaN  NaN

                A10   A11          A12    A13          A14  A15      A16
count    552.000000   552  552.000000    552   552.000000  552      552
unique          NaN     2         NaN      2          NaN    3        2
top             NaN  True         NaN  False          NaN    g  Failure
freq            NaN   314         NaN    305          NaN  496      297
```

```
mean      2.398678   NaN    2.614130    NaN   183.541667  NaN      NaN
std       3.551266   NaN    5.161073    NaN   182.638203  NaN      NaN
min       0.000000   NaN    0.000000    NaN     0.000000  NaN      NaN
25%       0.165000   NaN    0.000000    NaN    60.000000  NaN      NaN
50%       1.000000   NaN    0.000000    NaN   153.000000  NaN      NaN
75%       3.000000   NaN    3.000000    NaN   280.000000  NaN      NaN
max      28.500000   NaN   67.000000    NaN  2000.000000  NaN      NaN
```

[19]:
```
import numpy as np
from sklearn.preprocessing import LabelEncoder

#Since python machine learning algorithm do not accept string values
le = LabelEncoder()

dataset['A1'] = le.fit_transform(dataset['A1'])
dataset['A3'] = le.fit_transform(dataset['A3'])
dataset['A4'] = le.fit_transform(dataset['A4'])
dataset['A6'] = le.fit_transform(dataset['A6'])
dataset['A8'] = le.fit_transform(dataset['A8'])
dataset['A9'] = le.fit_transform(dataset['A9'])
dataset['A11'] = le.fit_transform(dataset['A11'])
dataset['A13'] = le.fit_transform(dataset['A13'])
dataset['A15'] = le.fit_transform(dataset['A15'])
```

[20]:
```
dataset
```

[20]:
```
     A1     A2  A3  A4      A5  A6      A7  A8  A9    A10  A11   A12  A13  \
0     2  30.83   2   1   0.000  13     0.0   1   8  1.250    1   1.0    0
1     1  58.67   2   1   4.460  11   560.0   1   4  3.040    1   6.0    0
2     1  24.50   2   1   0.500  11   824.0   0   4  1.500    1   0.0    0
3     2  27.83   2   1   1.540  13     3.0   1   8  3.750    1   5.0    1
4     2  25.00   2   1  11.250   2  1208.0   1   8  2.500    1  17.0    0
..   ..    ...  ..  ..     ...  ..     ...  ..  ..    ...  ...   ...  ...
547   2  39.17   2   1   1.625   2  4700.0   1   8  1.500    1  10.0    0
548   2  39.08   2   1   6.000  10  1097.0   1   8  1.290    1   5.0    1
549   2  31.67   2   1   0.830  14  3290.0   1   8  1.335    1   8.0    1
550   2  41.00   2   1   0.040   5     0.0   1   8  0.040    0   1.0    0
551   2  48.50   2   1   4.250  10     0.0   0   8  0.125    1   0.0    1

       A14  A15      A16
0    202.0    0  Success
1     43.0    0  Success
2    280.0    0  Success
3    100.0    0  Success
4    200.0    0  Success
..     ...  ...      ...
547  186.0    0  Success
```

```
548  108.0    0  Success
549  303.0    0  Success
550  560.0    2  Success
551  225.0    0  Success

[552 rows x 16 columns]
```

[21]:
```
#In above it has encoded the unique values of each column with a unique number
```

[22]:
```
print(dataset.describe())
```

```
                A1          A2          A3          A4          A5          A6  \
count   552.000000  552.000000  552.000000  552.000000  552.000000  552.000000
mean      1.672101   31.397373    2.217391    1.467391    4.884384    6.818841
std       0.499819   12.831325    0.470380    0.857774    5.086809    4.315566
min       0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
25%       1.000000   22.580000    2.000000    1.000000    1.083750    2.000000
50%       2.000000   28.210000    2.000000    1.000000    2.750000    7.000000
75%       2.000000   38.960000    2.000000    1.000000    7.551250   11.000000
max       2.000000   80.250000    3.000000    3.000000   28.000000   14.000000

                 A7          A8          A9         A10         A11  \
count     552.000000  552.000000  552.000000  552.000000  552.000000
mean     1100.827899    0.445652    5.905797    2.398678    0.568841
std      5628.306468    0.497488    2.629272    3.551266    0.495687
min         0.000000    0.000000    0.000000    0.000000    0.000000
25%         0.000000    0.000000    4.000000    0.165000    0.000000
50%         5.000000    0.000000    8.000000    1.000000    1.000000
75%       456.500000    1.000000    8.000000    3.000000    1.000000
max    100000.000000    1.000000    9.000000   28.500000    1.000000

              A12         A13          A14         A15
count  552.000000  552.000000   552.000000  552.000000
mean     2.614130    0.447464   183.541667    0.192029
std      5.161073    0.497683   182.638203    0.580451
min      0.000000    0.000000     0.000000    0.000000
25%      0.000000    0.000000    60.000000    0.000000
50%      0.000000    0.000000   153.000000    0.000000
75%      3.000000    1.000000   280.000000    0.000000
max     67.000000    1.000000  2000.000000    2.000000
```

[23]:
```
from sklearn.model_selection import train_test_split

X = dataset.values[:,0:15]
Y = dataset.values[:,15]

X_train, X_test, Y_train, Y_test = train_test_split(X, Y,random_state=0)
```

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

[24]:
```
#Let's train the machine learning algorithms with the dataset
#Then find the model with highest accuracy
```

[25]:
```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
logreg.fit(X_train, Y_train)
print('Accuracy of Logistic regression classifier on training set: {:.2f}'
     .format(logreg.score(X_train, Y_train)))
print('Accuracy of Logistic regression classifier on test set: {:.2f}'
     .format(logreg.score(X_test, Y_test)))
```

```
Accuracy of Logistic regression classifier on training set: 0.85
Accuracy of Logistic regression classifier on test set: 0.83
```

[26]:
```
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier().fit(X_train, Y_train)
print('Accuracy of Decision Tree classifier on training set: {:.2f}'
     .format(clf.score(X_train, Y_train)))
print('Accuracy of Decision Tree classifier on test set: {:.2f}'
     .format(clf.score(X_test, Y_test)))
```

```
Accuracy of Decision Tree classifier on training set: 1.00
Accuracy of Decision Tree classifier on test set: 0.80
```

[27]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(X_train, Y_train)
print('Accuracy of K-NN classifier on training set: {:.2f}'
     .format(knn.score(X_train, Y_train)))
print('Accuracy of K-NN classifier on test set: {:.2f}'
     .format(knn.score(X_test, Y_test)))
```

```
Accuracy of K-NN classifier on training set: 0.87
Accuracy of K-NN classifier on test set: 0.84
```

[28]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
lda = LinearDiscriminantAnalysis()
lda.fit(X_train, Y_train)
print('Accuracy of LDA classifier on training set: {:.2f}'
     .format(lda.score(X_train, Y_train)))
print('Accuracy of LDA classifier on test set: {:.2f}'
```

```
     .format(lda.score(X_test, Y_test)))
```

```
Accuracy of LDA classifier on training set: 0.85
Accuracy of LDA classifier on test set: 0.84
```

[29]:
```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X_train, Y_train)
print('Accuracy of GNB classifier on training set: {:.2f}'
     .format(gnb.score(X_train, Y_train)))
print('Accuracy of GNB classifier on test set: {:.2f}'
     .format(gnb.score(X_test, Y_test)))
```

```
Accuracy of GNB classifier on training set: 0.83
Accuracy of GNB classifier on test set: 0.83
```

[30]:
```
#Support Vector Machine algorithm
from sklearn.svm import SVC
svm = SVC()
svm.fit(X_train, Y_train)
print('Accuracy of SVM classifier on training set: {:.2f}'
     .format(svm.score(X_train, Y_train)))
print('Accuracy of SVM classifier on test set: {:.2f}'
     .format(svm.score(X_test, Y_test)))
```

```
Accuracy of SVM classifier on training set: 0.87
Accuracy of SVM classifier on test set: 0.86
```

[31]:
```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
pred = logreg.predict(X_test)
print(confusion_matrix(Y_test, pred))
print(classification_report(Y_test, pred))
```

```
[[64 17]
 [ 6 51]]
              precision    recall  f1-score   support

     Failure       0.91      0.79      0.85        81
     Success       0.75      0.89      0.82        57

    accuracy                           0.83       138
   macro avg       0.83      0.84      0.83       138
weighted avg       0.85      0.83      0.83       138
```

[32]:
```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
pred = clf.predict(X_test)
print(confusion_matrix(Y_test, pred))
print(classification_report(Y_test, pred))
```

```
[[61 20]
 [ 8 49]]
              precision    recall  f1-score   support

     Failure       0.88      0.75      0.81        81
     Success       0.71      0.86      0.78        57

    accuracy                           0.80       138
   macro avg       0.80      0.81      0.80       138
weighted avg       0.81      0.80      0.80       138
```

[33]:
```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
pred = knn.predict(X_test)
print(confusion_matrix(Y_test, pred))
print(classification_report(Y_test, pred))
```

```
[[68 13]
 [ 9 48]]
              precision    recall  f1-score   support

     Failure       0.88      0.84      0.86        81
     Success       0.79      0.84      0.81        57

    accuracy                           0.84       138
   macro avg       0.84      0.84      0.84       138
weighted avg       0.84      0.84      0.84       138
```

[34]:
```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
pred = lda.predict(X_test)
print(confusion_matrix(Y_test, pred))
print(classification_report(Y_test, pred))
```

```
[[62 19]
 [ 3 54]]
              precision    recall  f1-score   support

     Failure       0.95      0.77      0.85        81
     Success       0.74      0.95      0.83        57
```

```
    accuracy                           0.84       138
   macro avg       0.85      0.86      0.84       138
weighted avg       0.87      0.84      0.84       138
```

[35]:
```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
pred = gnb.predict(X_test)
print(confusion_matrix(Y_test, pred))
print(classification_report(Y_test, pred))
```

```
[[74  7]
 [16 41]]
              precision    recall  f1-score   support

     Failure       0.82      0.91      0.87        81
     Success       0.85      0.72      0.78        57

    accuracy                           0.83       138
   macro avg       0.84      0.82      0.82       138
weighted avg       0.84      0.83      0.83       138
```

[36]:
```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
pred = svm.predict(X_test)
print(confusion_matrix(Y_test, pred))
print(classification_report(Y_test, pred))
```

```
[[66 15]
 [ 5 52]]
              precision    recall  f1-score   support

     Failure       0.93      0.81      0.87        81
     Success       0.78      0.91      0.84        57

    accuracy                           0.86       138
   macro avg       0.85      0.86      0.85       138
weighted avg       0.87      0.86      0.86       138
```

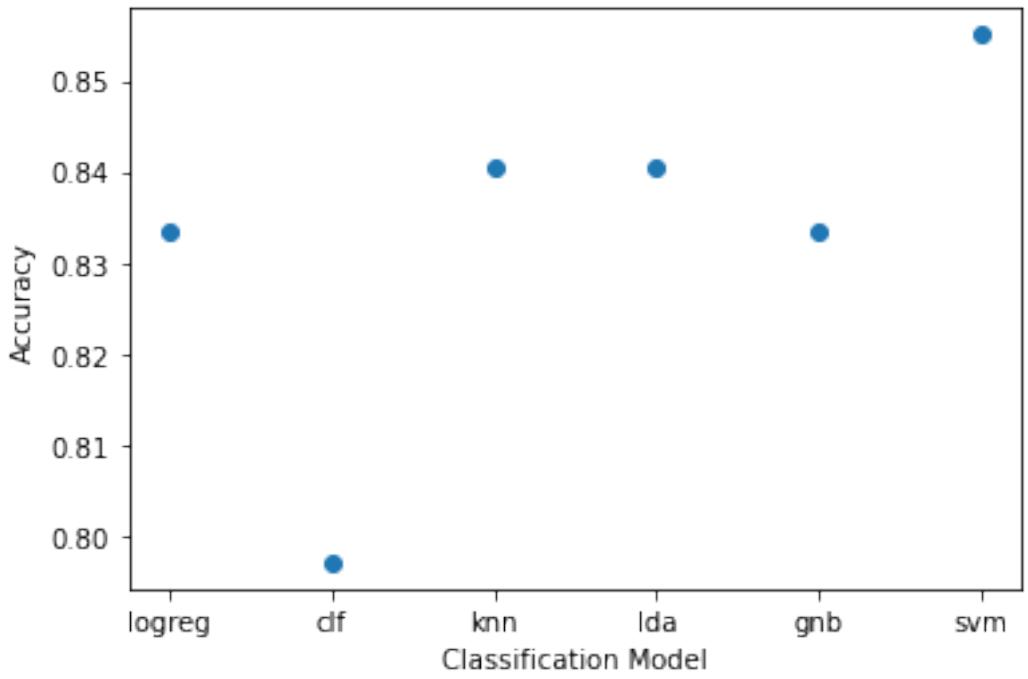
[38]:
```
model = ['logreg','clf','knn','lda','gnb','svm']
scores = []

scores.append(logreg.score(X_test, Y_test))
scores.append(clf.score(X_test, Y_test))
scores.append(knn.score(X_test, Y_test))
```

```
scores.append(lda.score(X_test, Y_test))
scores.append(gnb.score(X_test, Y_test))
scores.append(svm.score(X_test, Y_test))

plt.figure()
plt.xlabel('Classification Model')
plt.ylabel('Accuracy')
plt.scatter(model, scores)
```

[38]: <matplotlib.collections.PathCollection at 0x21ebbddf370>



[39]:
```
testset = pd.read_csv('E:/University Works/3rd Year/Semester 6/CO 544 - Machine␣
↪Learning and Data Mining/Project/new/testdata.csv')
testset.head()
```

[39]:

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	\
0	b	32.67	y	p	9.00	w	0	False	h	5.25	True	0	True	154	
1	a	28.08	y	p	15.00	e	13212	False	z	0.00	True	0	False	0	
2	b	73.42	u	g	17.75	ff	0	False	ff	0.00	True	0	True	0	
3	b	64.08	u	g	20.00	x	1000	True	h	17.50	True	9	True	0	
4	b	51.58	u	g	15.00	c	0	True	v	8.50	True	9	False	0	

	A15	A16
0	g	NaN

```
1   g  NaN
2   g  NaN
3   g  NaN
4   g  NaN
```

[40]:
```
print ("Dataset Length: ", len(testset))
print ("Dataset Shape: ", testset.shape)
```

```
Dataset Length:  138
Dataset Shape:  (138, 16)
```

[41]:
```
tf = pd.DataFrame(testset)
```

[42]:
```
testset[['A2','A5','A7','A10','A12','A14']] =␣
↪testset[['A2','A5','A7','A10','A12','A14']].replace('?',0)
# to change use .astype()
testset['A2'] = testset.A2.astype(float)
testset['A5'] = testset.A5.astype(float)
testset['A7'] = testset.A7.astype(float)
testset['A10'] = testset.A10.astype(float)
testset['A12'] = testset.A12.astype(float)
testset['A14'] = testset.A14.astype(float)
```

[43]:
```
tf['A4'] = le.fit_transform(tf['A4'])
tf['A1'] = le.fit_transform(tf['A1'])
tf['A3'] = le.fit_transform(tf['A3'])
tf['A6'] = le.fit_transform(tf['A6'])
tf['A8'] = le.fit_transform(tf['A8'])
tf['A9'] = le.fit_transform(tf['A9'])
tf['A11'] = le.fit_transform(tf['A11'])
tf['A13'] = le.fit_transform(tf['A13'])
tf['A15'] = le.fit_transform(tf['A15'])
```

[44]:
```
test_scaler = scaler.transform(tf.values[:,0:15])
```

[46]:
```
test_pred = svm.predict(test_scaler)
print("Predicted values:")
print(test_pred)
```

```
Predicted values:
['Success' 'Success' 'Success' 'Success' 'Success' 'Success' 'Success'
 'Success' 'Success' 'Failure' 'Failure' 'Success' 'Success' 'Success'
 'Success' 'Failure' 'Success' 'Success' 'Failure' 'Failure' 'Failure'
 'Success' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure'
 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure'
 'Failure' 'Success' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure'
 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure'
```

```
'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure'
'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure'
'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure'
'Success' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure'
'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure'
'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure'
'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure'
'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure'
'Failure' 'Failure' 'Success' 'Success' 'Success' 'Success' 'Success'
'Success' 'Failure' 'Success' 'Success' 'Failure' 'Success' 'Success'
'Success' 'Success' 'Success' 'Success' 'Success' 'Success' 'Success'
'Success' 'Success' 'Success' 'Success' 'Success' 'Success' 'Success'
'Success' 'Success' 'Success' 'Success' 'Success']
```

[47]:
```
tf['A16'] = test_pred
#Final prediction on the test data set
tf
```

[47]:

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16
0	2	32.67	2	2	9.00	13	0.0	0	3	5.25	1	0.0	1	154.0	0	Success
1	1	28.08	2	2	15.00	5	13212.0	0	7	0.00	1	0.0	0	0.0	0	Success
2	2	73.42	1	1	17.75	6	0.0	0	2	0.00	1	0.0	1	0.0	0	Success
3	2	64.08	1	1	20.00	14	1000.0	1	3	17.50	1	9.0	1	0.0	0	Success
4	2	51.58	1	1	15.00	2	0.0	1	6	8.50	1	9.0	0	0.0	0	Success
..	..	…	..	..	…	..	…	..	..	…	…	…	…	…	…	…
133	1	30.67	1	1	12.00	2	19.0	1	6	2.00	1	1.0	0	220.0	0	Success
134	2	21.00	2	2	4.79	13	300.0	1	6	2.25	1	1.0	1	80.0	0	Success
135	2	13.75	2	2	4.00	13	1000.0	1	6	1.75	1	2.0	1	120.0	0	Success
136	1	46.00	1	1	4.00	8	960.0	0	4	0.00	1	0.0	0	100.0	0	Success
137	1	44.33	1	1	0.00	2	0.0	0	6	2.50	1	0.0	0	0.0	0	Success

[138 rows x 16 columns]

[48]:
```
#This is to show the final test without any encodings
test_final = pd.read_csv('E:/University Works/3rd Year/Semester 6/CO 544 -␣
↪Machine Learning and Data Mining/Project/new/testdata.csv')
test_final['A16'] = test_pred
```

[49]:
```
test_final_frame = pd.DataFrame(test_final)
```

[50]:
```
#Final prediction on the test set
test_final_frame
```

[50]:
```
    A1     A2 A3 A4     A5  A6     A7     A8  A9    A10   A11  A12    A13  \
0    b  32.67  y  p   9.00   w      0  False   h   5.25  True    0   True
1    a  28.08  y  p  15.00   e  13212  False   z   0.00  True    0  False
2    b  73.42  u  g  17.75  ff      0  False  ff   0.00  True    0   True
3    b  64.08  u  g  20.00   x   1000   True   h  17.50  True    9   True
4    b  51.58  u  g  15.00   c      0   True   v   8.50  True    9  False
..  ..    ... .. ..    ...  ..    ...    ...  ..    ...   ...  ...    ...
133  a  30.67  u  g  12.00   c     19   True   v   2.00  True    1  False
134  b     21  y  p   4.79   w    300   True   v   2.25  True    1   True
135  b  13.75  y  p   4.00   w   1000   True   v   1.75  True    2   True
136  a     46  u  g   4.00   j    960  False   j   0.00  True    0  False
137  a  44.33  u  g   0.00   c      0  False   v   2.50  True    0  False

     A14 A15      A16
0    154   g  Success
1      0   g  Success
2      0   g  Success
3      0   g  Success
4      0   g  Success
..   ...  ..      ...
133  220   g  Success
134   80   g  Success
135  120   g  Success
136  100   g  Success
137    0   g  Success

[138 rows x 16 columns]
```

[51]:
```
test_final_frame.to_csv('E:/University Works/3rd Year/Semester 6/CO 544 -␣
↪Machine Learning and Data Mining/Project/new/testresultsnew.csv',sep=',')
```

[ ]:
```

```