In [48]:
```
# Load required libraries
from sklearn import datasets
from sklearn.linear_model import Perceptron
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,confusion_matrix, precision_recall_fscore_support
from sklearn.metrics import classification_report,average_precision_score,precision_recall_curve
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.utils.fixes import signature
import warnings
warnings.filterwarnings('ignore')
```

In [49]:
```
wine_data = pd.read_csv("wine.csv")
```

In [50]:
```
wine_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6463 entries, 0 to 6462
Data columns (total 14 columns):
type                    6463 non-null object
fixed acidity           6463 non-null float64
volatile acidity        6463 non-null float64
citric acid             6463 non-null float64
residual sugar          6463 non-null float64
chlorides               6463 non-null float64
free sulfur dioxide     6463 non-null float64
total sulfur dioxide    6463 non-null float64
density                 6463 non-null float64
pH                      6463 non-null float64
sulphates               6463 non-null float64
alcohol                 6463 non-null float64
quality                 6463 non-null int64
good/bad                6463 non-null object
dtypes: float64(11), int64(1), object(2)
memory usage: 707.0+ KB
```

In [51]:
```
wine_data.head()
```

Out[51]:

	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	good/bad
0	white	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6	bad
1	white	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6	bad
2	white	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6	bad
3	white	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6	bad
4	white	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6	bad

In [52]:
```
wine_data.describe()
```

Out[52]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulpha
count	6463.000000	6463.000000	6463.000000	6463.000000	6463.000000	6463.000000	6463.000000	6463.000000	6463.000000	6463.00C
mean	7.217755	0.339589	0.318758	5.443958	0.056056	30.516865	115.694492	0.994698	3.218332	0.531
std	1.297913	0.164639	0.145252	4.756852	0.035076	17.758815	56.526736	0.003001	0.160650	0.148
min	3.800000	0.080000	0.000000	0.600000	0.009000	1.000000	6.000000	0.987110	2.720000	0.22C
25%	6.400000	0.230000	0.250000	1.800000	0.038000	17.000000	77.000000	0.992330	3.110000	0.43C
50%	7.000000	0.290000	0.310000	3.000000	0.047000	29.000000	118.000000	0.994890	3.210000	0.51C
75%	7.700000	0.400000	0.390000	8.100000	0.065000	41.000000	156.000000	0.997000	3.320000	0.60C
max	15.900000	1.580000	1.660000	65.800000	0.611000	289.000000	440.000000	1.038980	4.010000	2.00C

In [53]:
```
y=wine_data['quality']
y=y.to_frame()
y.head()
```

Out[53]:

	quality
0	6
1	6
2	6
3	6
4	6

In [54]:
```
X=wine_data
```

In [55]:
```
X.head()
```

Out[55]:

	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	good/bad
0	white	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6	bad
1	white	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6	bad
2	white	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6	bad
3	white	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6	bad
4	white	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6	bad

In [56]:
```
#Applying Train,Test Split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=32)
```

In [57]:
```
pd.set_option('mode.chained_assignment', None)
```

In [58]:
```
#assigning 1 to yes and 0 to no in wine (X_train)
combine=[X_train,X_test]
winemapping={'red':1,'white': 0}
for dt in combine:
   dt['type']=wine_data['type'].map(winemapping)
X_train.head()
```

Out[58]:

	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	good/bad
4373	0	6.6	0.24	0.22	12.3	0.051	35.0	146.0	0.99676	3.10	0.67	9.4	5	bad
2047	0	6.5	0.19	0.26	5.2	0.040	31.0	140.0	0.99500	3.26	0.68	9.5	6	bad
753	0	6.1	0.27	0.30	16.7	0.039	49.0	172.0	0.99985	3.40	0.45	9.4	5	bad
4538	0	7.0	0.23	0.35	1.4	0.036	31.0	113.0	0.99120	3.16	0.48	10.8	7	good
3563	0	6.8	0.19	0.71	17.5	0.042	21.0	114.0	0.99784	2.85	0.50	9.5	6	bad

In [59]:
```
#assigning 1 to yes and 0 to no in wine (y_train)
combine=[X_train,X_test]
winemapping={'good':1,'bad': 0}
for dt in combine:
   dt['good/bad']=wine_data['good/bad'].map(winemapping)
X_train.head()
```

Out[59]:

	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	good/bad
4373	0	6.6	0.24	0.22	12.3	0.051	35.0	146.0	0.99676	3.10	0.67	9.4	5	0
2047	0	6.5	0.19	0.26	5.2	0.040	31.0	140.0	0.99500	3.26	0.68	9.5	6	0
753	0	6.1	0.27	0.30	16.7	0.039	49.0	172.0	0.99985	3.40	0.45	9.4	5	0
4538	0	7.0	0.23	0.35	1.4	0.036	31.0	113.0	0.99120	3.16	0.48	10.8	7	1
3563	0	6.8	0.19	0.71	17.5	0.042	21.0	114.0	0.99784	2.85	0.50	9.5	6	0

In [60]:
```
X_final_train = X_train[['type', 'fixed acidity', 'volatile acidity', 'citric acid',
       'residual sugar', 'chlorides', 'free sulfur dioxide',
       'total sulfur dioxide', 'density', 'pH', 'sulphates', 'alcohol',
        'good/bad']]
X_final_test = X_test[['type', 'fixed acidity', 'volatile acidity', 'citric acid',
       'residual sugar', 'chlorides', 'free sulfur dioxide',
       'total sulfur dioxide', 'density', 'pH', 'sulphates', 'alcohol',
        'good/bad']]
```

In [61]:
```
X_final_train.columns
```

Out[61]:
```
Index(['type', 'fixed acidity', 'volatile acidity', 'citric acid',
       'residual sugar', 'chlorides', 'free sulfur dioxide',
       'total sulfur dioxide', 'density', 'pH', 'sulphates', 'alcohol',
       'good/bad'],
      dtype='object')
```

In [62]:
```
# Create a perceptron object with the parameters: 40 iterations (epochs) over the data, and a learnin
g rate of 0.1
ppn = Perceptron(n_iter=40, eta0=0.1, random_state=0)

# Train the perceptron
ppn.fit(X_final_train, y_train)
```

Out[62]:
```
Perceptron(alpha=0.0001, class_weight=None, early_stopping=False, eta0=0.1,
      fit_intercept=True, max_iter=None, n_iter=40, n_iter_no_change=5,
      n_jobs=None, penalty=None, random_state=0, shuffle=True, tol=None,
      validation_fraction=0.1, verbose=0, warm_start=False)
```

In [63]:
```
X_final_train.columns
```

Out[63]:
```
Index(['type', 'fixed acidity', 'volatile acidity', 'citric acid',
       'residual sugar', 'chlorides', 'free sulfur dioxide',
       'total sulfur dioxide', 'density', 'pH', 'sulphates', 'alcohol',
       'good/bad'],
      dtype='object')
```

In [64]:
```
# Apply the trained perceptron on the X data to make predicts for the y test data
y_pred = ppn.predict(X_final_test)
```

In [65]:
```
y_pred
```

Out[65]:
```
array([5, 5, 5, ..., 5, 5, 6])
```

In [66]:
```
y_test.head()
```

Out[66]:

	quality
2558	7
5380	5
924	5
4124	5
4386	7

In [67]:
```
#View the accuracy of the model, which is: 1 - (observations predicted wrong / total observations)
 #Accuracy: The amount of correct classifications / the total amount of classifications.
 #The train accuracy: The accuracy of a model on examples it was constructed on.
  #The test accuracy is the accuracy of a model on examples it hasn't seen.
accuracy_test_ppn=round(ppn.score(X_final_test,y_test)*100,2)
accuracy_train_ppn=round(ppn.score(X_final_train,y_train)*100,2)
accuracy_ppn=round(accuracy_score(y_test, y_pred)*100,2)
print('Training accuracy of perceptron',accuracy_train_ppn)
print('Testing accuracy of perceptron',accuracy_test_ppn)
print('Accuracy of Perceptron:',accuracy_ppn)
```

```
Training accuracy of perceptron 63.99
Testing accuracy of perceptron 64.21
Accuracy of Perceptron: 64.21
```

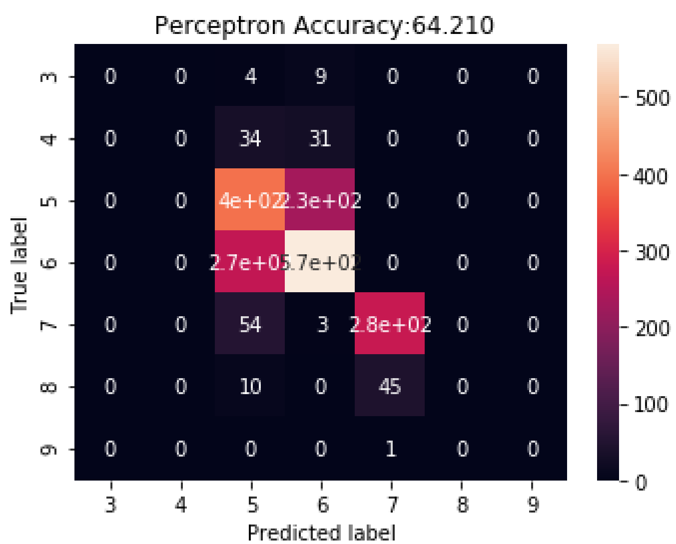
In [68]:
```
#Confusion Matrix for Perceptron
cm = confusion_matrix(y_test, y_pred)
```

In [69]:
```
cm
```

Out[69]:
```
array([[  0,   0,   4,   9,   0,   0,   0],
       [  0,   0,  34,  31,   0,   0,   0],
       [  0,   0, 398, 232,   0,   0,   0],
       [  0,   0, 271, 568,   0,   0,   0],
       [  0,   0,  54,   3, 279,   0,   0],
       [  0,   0,  10,   0,  45,   0,   0],
       [  0,   0,   0,   0,   1,   0,   0]])
```

In [70]:
```
cm_df = pd.DataFrame(cm,
                     index = ['3','4','5','6','7','8','9'],
                     columns = ['3','4','5','6','7','8','9'])
```

In [71]:
```
plt.figure(figsize=(5.5,4))
sns.heatmap(cm_df, annot=True)
plt.title(' Perceptron Accuracy:{0:.3f}'.format(accuracy_test_ppn))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```



In [72]:
```
target_names = ['3','4','5','6','7','8','9']
print(classification_report(y_test, y_pred, target_names=target_names))
```

```
              precision    recall  f1-score   support

           3       0.00      0.00      0.00        13
           4       0.00      0.00      0.00        65
           5       0.52      0.63      0.57       630
           6       0.67      0.68      0.68       839
           7       0.86      0.83      0.84       336
           8       0.00      0.00      0.00        55
           9       0.00      0.00      0.00         1

   micro avg       0.64      0.64      0.64      1939
   macro avg       0.29      0.31      0.30      1939
weighted avg       0.61      0.64      0.62      1939
```

In [73]:
```
#SVM implementation with same dataset
svm_clf = SVC()
```

In [74]:
```
svm_clf.fit(X_final_train,y_train)
```

Out[74]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
```

In [75]:
```
svm_pred = svm_clf.predict(X_final_test)
```

In [76]:
```
svm_pred
```

Out[76]:
```
array([7, 6, 6, ..., 5, 6, 6])
```

In [77]:
```
y_test.head()
```

Out[77]:

	quality
2558	7
5380	5
924	5
4124	5
4386	7

In [78]:
```
accuracy_test_svm=round(svm_clf.score(X_final_test,y_test)*100,2)
accuracy_train_svm=round(svm_clf.score(X_final_train,y_train)*100,2)
accuracy_svm=round(accuracy_score(y_test, svm_pred)*100,2)
print('Training accuracy of SVM',accuracy_train_svm)
print('Testing accuracy of SVM',accuracy_test_svm)
print('Accuracy of SVM classifier:',accuracy_svm)
```

```
Training accuracy of SVM 82.76
Testing accuracy of SVM 57.25
Accuracy of SVM classifier: 57.25
```

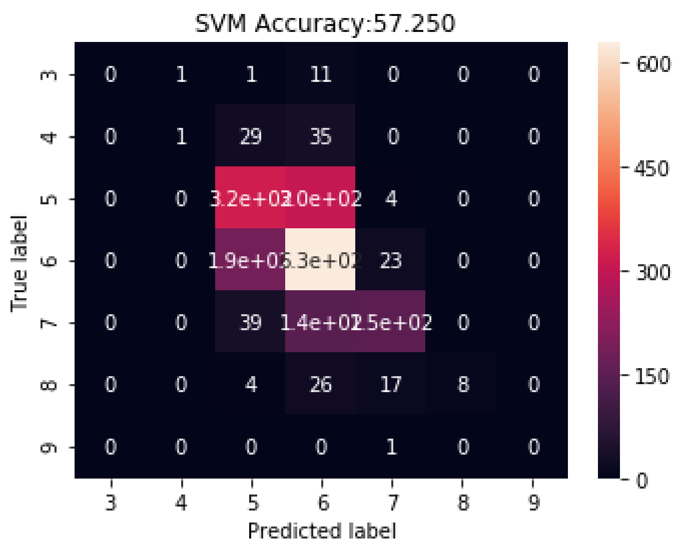
In [79]:
```
#Confusion Matrix for SVM
cm = confusion_matrix(y_test, svm_pred)
```

In [80]:
```
cm
```

Out[80]:
```
array([[  0,   1,   1,  11,   0,   0,   0],
       [  0,   1,  29,  35,   0,   0,   0],
       [  0,   0, 321, 305,   4,   0,   0],
       [  0,   0, 188, 628,  23,   0,   0],
       [  0,   0,  39, 145, 152,   0,   0],
       [  0,   0,   4,  26,  17,   8,   0],
       [  0,   0,   0,   0,   1,   0,   0]])
```

In [81]:
```
cm_df = pd.DataFrame(cm,
                     index = ['3','4','5','6','7','8','9'],
                     columns = ['3','4','5','6','7','8','9'])
```

In [82]:
```
plt.figure(figsize=(5.5,4))
sns.heatmap(cm_df, annot=True)
plt.title(' SVM Accuracy:{0:.3f}'.format(accuracy_test_svm))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```



In [83]:
```
target_names = ['3','4','5','6','7','8','9']
print(classification_report(y_test, y_pred, target_names=target_names))
```

```
              precision    recall  f1-score   support

           3       0.00      0.00      0.00        13
           4       0.00      0.00      0.00        65
           5       0.52      0.63      0.57       630
           6       0.67      0.68      0.68       839
           7       0.86      0.83      0.84       336
           8       0.00      0.00      0.00        55
           9       0.00      0.00      0.00         1

   micro avg       0.64      0.64      0.64      1939
   macro avg       0.29      0.31      0.30      1939
weighted avg       0.61      0.64      0.62      1939
```

In [84]:
```
#Logistic Regression for same dataset
log_clf = LogisticRegression(random_state=0, solver='lbfgs',multi_class='multinomial')
```

In [85]:
```
log_clf.fit(X_final_train,y_train)
```

Out[85]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='multinomial',
          n_jobs=None, penalty='l2', random_state=0, solver='lbfgs',
          tol=0.0001, verbose=0, warm_start=False)
```

In [86]:
```
log_pred = log_clf.predict(X_final_test)
```

In [87]:
```
log_pred
```

Out[87]:
```
array([5, 5, 5, ..., 6, 5, 6])
```

In [88]:
```
y_test.head()
```

Out[88]:

	quality
2558	7
5380	5
924	5
4124	5
4386	7

In [89]:
```
accuracy_test_log=round(log_clf.score(X_final_test,y_test)*100,2)
accuracy_train_log=round(log_clf.score(X_final_train,y_train)*100,2)
accuracy_log=round(accuracy_score(y_test, log_pred)*100,2)
print('Training accuracy of Logistic regression',accuracy_train_log)
print('Testing accuracy of Logistic regression',accuracy_test_log)
print('Accuracy of Logistic regression:',accuracy_log)
```

```
Training accuracy of Logistic regression 58.93
Testing accuracy of Logistic regression 60.5
Accuracy of Logistic regression: 60.5
```

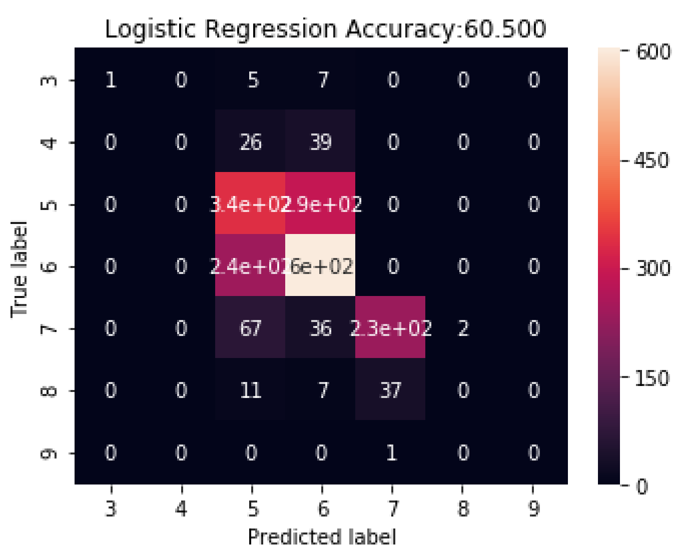
In [90]:
```
#Confusion Matrix for Logistic Regression
cm = confusion_matrix(y_test, log_pred)
```

In [91]:
```
cm
```

Out[91]:
```
array([[  1,   0,   5,   7,   0,   0,   0],
       [  0,   0,  26,  39,   0,   0,   0],
       [  0,   0, 338, 292,   0,   0,   0],
       [  0,   0, 236, 603,   0,   0,   0],
       [  0,   0,  67,  36, 231,   2,   0],
       [  0,   0,  11,   7,  37,   0,   0],
       [  0,   0,   0,   0,   1,   0,   0]])
```

In [92]:
```
cm_df = pd.DataFrame(cm,
                     index = ['3','4','5','6','7','8','9'],
                     columns = ['3','4','5','6','7','8','9'])
```

In [93]:
```
plt.figure(figsize=(5.5,4))
sns.heatmap(cm_df, annot=True)
plt.title(' Logistic Regression Accuracy:{0:.3f}'.format(accuracy_test_log))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```



In [94]:
```
target_names = ['3','4','5','6','7','8','9']
print(classification_report(y_test, y_pred, target_names=target_names))
```

```
              precision    recall  f1-score   support

           3       0.00      0.00      0.00        13
           4       0.00      0.00      0.00        65
           5       0.52      0.63      0.57       630
           6       0.67      0.68      0.68       839
           7       0.86      0.83      0.84       336
           8       0.00      0.00      0.00        55
           9       0.00      0.00      0.00         1

   micro avg       0.64      0.64      0.64      1939
   macro avg       0.29      0.31      0.30      1939
weighted avg       0.61      0.64      0.62      1939
```

In [ ]: