In [1]:

```
pip install feature-engine
```

```
Collecting feature-engine
  Downloading feature_engine-1.6.2-py2.py3-none-any.whl (328 kB)
     -------------------------------------- 328.9/328.9 kB 2.3 MB/s eta 0:00:00
Requirement already satisfied: pandas>=1.0.3 in c:\users\99210\anaconda3\lib\site-pac
kages (from feature-engine) (1.4.4)
Requirement already satisfied: scikit-learn>=1.0.0 in c:\users\99210\anaconda3\lib\si
te-packages (from feature-engine) (1.0.2)
Requirement already satisfied: numpy>=1.18.2 in c:\users\99210\anaconda3\lib\site-pac
kages (from feature-engine) (1.21.5)
Requirement already satisfied: scipy>=1.4.1 in c:\users\99210\anaconda3\lib\site-pack
ages (from feature-engine) (1.9.1)
Requirement already satisfied: statsmodels>=0.11.1 in c:\users\99210\anaconda3\lib\si
te-packages (from feature-engine) (0.13.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\99210\anaconda3\lib\site-pack
ages (from pandas>=1.0.3->feature-engine) (2022.1)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\99210\anaconda3\lib
\site-packages (from pandas>=1.0.3->feature-engine) (2.8.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\99210\anaconda3\lib\s
ite-packages (from scikit-learn>=1.0.0->feature-engine) (2.2.0)
Requirement already satisfied: joblib>=0.11 in c:\users\99210\anaconda3\lib\site-pack
ages (from scikit-learn>=1.0.0->feature-engine) (1.1.0)
Requirement already satisfied: patsy>=0.5.2 in c:\users\99210\anaconda3\lib\site-pack
ages (from statsmodels>=0.11.1->feature-engine) (0.5.2)
Requirement already satisfied: packaging>=21.3 in c:\users\99210\anaconda3\lib\site-p
ackages (from statsmodels>=0.11.1->feature-engine) (21.3)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in c:\users\99210\anaconda3\l
ib\site-packages (from packaging>=21.3->statsmodels>=0.11.1->feature-engine) (3.0.9)
Requirement already satisfied: six in c:\users\99210\anaconda3\lib\site-packages (fro
m patsy>=0.5.2->statsmodels>=0.11.1->feature-engine) (1.16.0)
Installing collected packages: feature-engine
Successfully installed feature-engine-1.6.2
Note: you may need to restart the kernel to use updated packages.
```

In [2]:

```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt

from feature_engine.outliers import Winsorizer
import feature_engine.transformation as vt

# for one hot encoding with sklearn
from sklearn.preprocessing import OneHotEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier

from collections import Counter
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
%matplotlib inline

# Ignoring Unnecessary warnings
import warnings
warnings.filterwarnings("ignore")
```

In [4]:

```
df = pd.read_csv("C:/Users/99210/Downloads/winedataset.csv")
```

In [5]:

```
df.head(10)
```

Out[5]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
5	7.4	0.66	0.00	1.8	0.075	13.0	40.0	0.9978	3.51	0.56	9.4	5
6	7.9	0.60	0.06	1.6	0.069	15.0	59.0	0.9964	3.30	0.46	9.4	5
7	7.3	0.65	0.00	1.2	0.065	15.0	21.0	0.9946	3.39	0.47	10.0	7
8	7.8	0.58	0.02	2.0	0.073	9.0	18.0	0.9968	3.36	0.57	9.5	7
9	7.5	0.50	0.36	6.1	0.071	17.0	102.0	0.9978	3.35	0.80	10.5	5

In [6]:

```
df.shape
```

Out[6]:

```
(1599, 12)
```

In [7]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

In [8]:

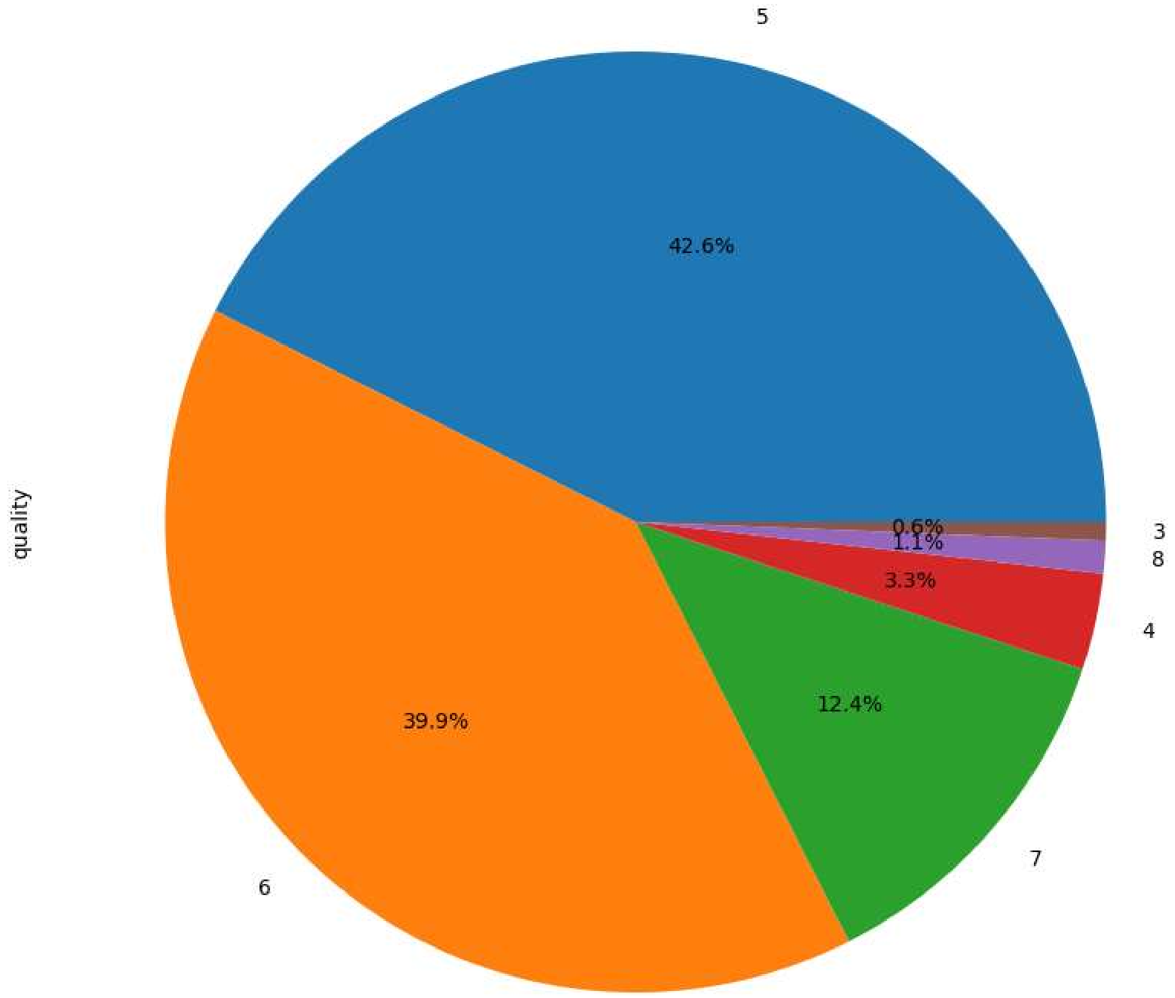
```
df.quality.unique()
```

Out[8]:

```
array([5, 6, 7, 4, 8, 3], dtype=int64)
```

In [9]:

```
plt.figure(1, figsize=(10,10))
df['quality'].value_counts().plot.pie(autopct="%1.1f%%")
plt.show()
```



In [10]:

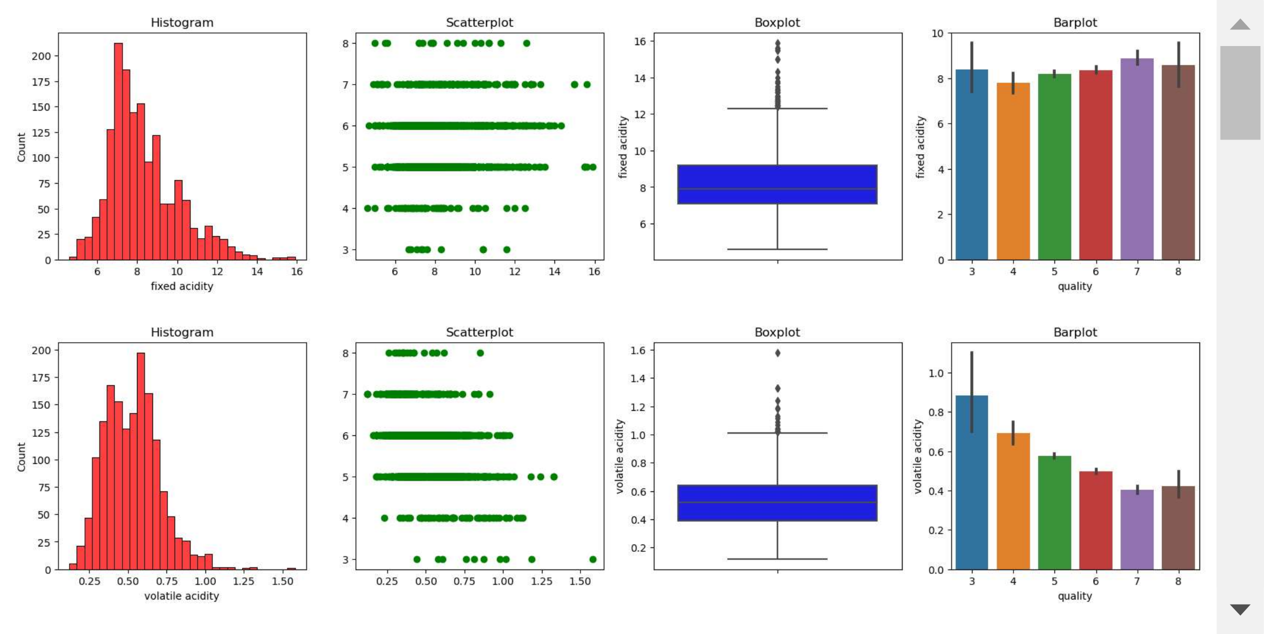
```
df.quality.value_counts(ascending=False)
```

Out[10]:

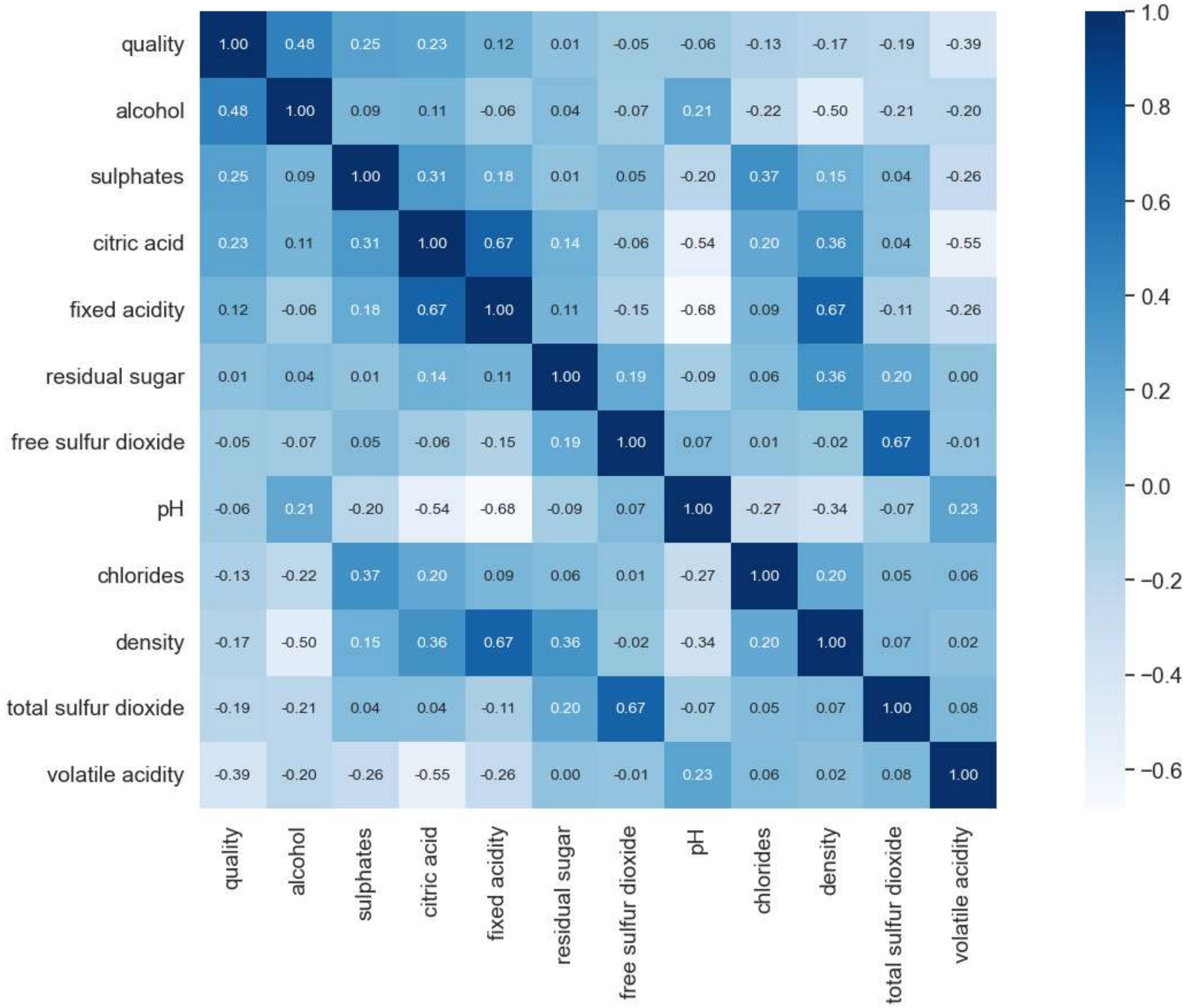
```
5    681
6    638
7    199
4     53
8     18
3     10
Name: quality, dtype: int64
```

In [12]:
```python
def diagnostic_plots(df, variable,target):
    # The function takes a dataframe (df) and
    # the variable of interest as arguments.

    # Define figure size.
    plt.figure(figsize=(20, 4))

    # histogram
    plt.subplot(1, 4, 1)
    sns.histplot(df[variable], bins=30,color = 'r')
    plt.title('Histogram')


    # scatterplot
    plt.subplot(1, 4, 2)
    plt.scatter(df[variable],df[target],color = 'g')
    plt.title('Scatterplot')


    # boxplot
    plt.subplot(1, 4, 3)
    sns.boxplot(y=df[variable],color = 'b')
    plt.title('Boxplot')

    # barplot
    plt.subplot(1, 4, 4)
    sns.barplot(x = target, y = variable, data = df)
    plt.title('Barplot')


    plt.show()
```

In [13]:
```python
for variable in df:
    diagnostic_plots(df,variable,'quality')
```



In [14]:
```
corr = df.corr()
plt.figure(figsize=(20, 9))
k = 12 #number of variables for heatmap
cols = corr.nlargest(k, 'quality')['quality'].index
cm = np.corrcoef(df[cols].values.T)
sns.set(font_scale=1.25)
hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f', annot_kws={'size':
plt.show()
```



	quality	alcohol	sulphates	citric acid	fixed acidity	residual sugar	free sulfur dioxide	pH	chlorides	density	total sulfur dioxide	volatile acidity
quality	1.00	0.48	0.25	0.23	0.12	0.01	-0.05	-0.06	-0.13	-0.17	-0.19	-0.39
alcohol	0.48	1.00	0.09	0.11	-0.06	0.04	-0.07	0.21	-0.22	-0.50	-0.21	-0.20
sulphates	0.25	0.09	1.00	0.31	0.18	0.01	0.05	-0.20	0.37	0.15	0.04	-0.26
citric acid	0.23	0.11	0.31	1.00	0.67	0.14	-0.06	-0.54	0.20	0.36	0.04	-0.55
fixed acidity	0.12	-0.06	0.18	0.67	1.00	0.11	-0.15	-0.68	0.09	0.67	-0.11	-0.26
residual sugar	0.01	0.04	0.01	0.14	0.11	1.00	0.19	-0.09	0.06	0.36	0.20	0.00
free sulfur dioxide	-0.05	-0.07	0.05	-0.06	-0.15	0.19	1.00	0.07	0.01	-0.02	0.67	-0.01
pH	-0.06	0.21	-0.20	-0.54	-0.68	-0.09	0.07	1.00	-0.27	-0.34	-0.07	0.23
chlorides	-0.13	-0.22	0.37	0.20	0.09	0.06	0.01	-0.27	1.00	0.20	0.05	0.06
density	-0.17	-0.50	0.15	0.36	0.67	0.36	-0.02	-0.34	0.20	1.00	0.07	0.02
total sulfur dioxide	-0.19	-0.21	0.04	0.04	-0.11	0.20	0.67	-0.07	0.05	0.07	1.00	0.08
volatile acidity	-0.39	-0.20	-0.26	-0.55	-0.26	0.00	-0.01	0.23	0.06	0.02	0.08	1.00

In [15]:

```
df.isnull().sum()
```

Out[15]:

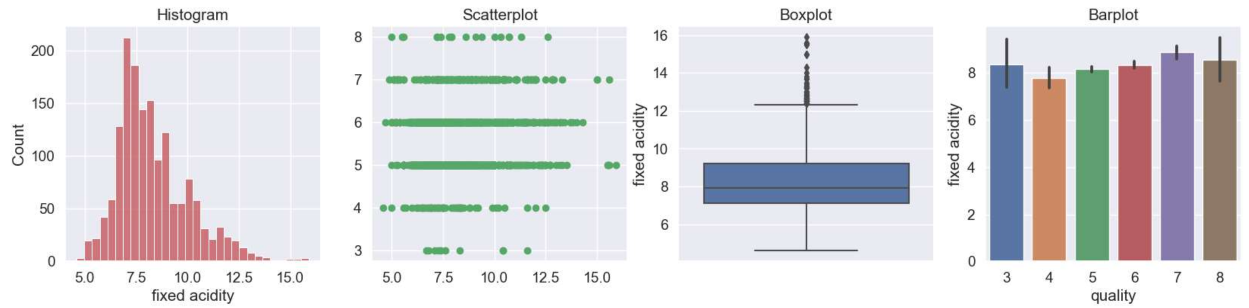
```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64
```

In [17]:

```
def detect_outliers(df,features):
    outlier_indices = []

    for c in features:
        # 1st quartile
        Q1 = np.percentile(df[c],25)
        # 3rd quartile
        Q3 = np.percentile(df[c],75)
        # IQR
        IQR = Q3 - Q1
        # Outlier step
        outlier_step = IQR * 1.5
        # detect outlier and their indeces
        outlier_list_col = df[(df[c] < Q1 - outlier_step) | (df[c] > Q3 + outlier_step
        outlier_indices.extend(outlier_list_col)

    outlier_indices = Counter(outlier_indices)
    multiple_outliers = list(i for i, v in outlier_indices.items() if v > 2)

    return multiple_outliers
```

In [18]:

```
diagnostic_plots(df,'fixed acidity','quality')
```



In [19]:

```
cols = ['fixed acidity', 'volatile acidity', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide',
       'sulphates', 'alcohol']
```

In [20]:
```
lt = vt.LogTransformer(variables = cols)

lt.fit(df)
```

Out[20]:
```
LogTransformer(variables=['fixed acidity', 'volatile acidity', 'residual sugar',
                          'chlorides', 'free sulfur dioxide',
                          'total sulfur dioxide', 'sulphates', 'alcohol'])
```

In [21]:
```
df = lt.transform(df)
```

In [22]:
```
bins = (2, 6.5, 8)
group_names = ['bad', 'good']
df['quality'] = pd.cut(df['quality'], bins = bins, labels = group_names)
```

In [25]:
```
encoder = LabelEncoder()
```

In [24]:
```
df['quality'] = encoder.fit_transform(df['quality'])
```

In [26]:
```
df['quality'].value_counts()
```

Out[26]:
```
0    1382
1     217
Name: quality, dtype: int64
```

In [27]:
```
df.head()
```

Out[27]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	2.001480	-0.356675	0.00	0.641854	-2.577022	2.397895	3.526361	0.9978	3.51	-0.579818	2.240710
1	2.054124	-0.127833	0.00	0.955511	-2.322788	3.218876	4.204693	0.9968	3.20	-0.385662	2.282382
2	2.054124	-0.274437	0.04	0.832909	-2.385967	2.708050	3.988984	0.9970	3.26	-0.430783	2.282382
3	2.415914	-1.272966	0.56	0.641854	-2.590267	2.833213	4.094345	0.9980	3.16	-0.544727	2.282382
4	2.001480	-0.356675	0.00	0.641854	-2.577022	2.397895	3.526361	0.9978	3.51	-0.579818	2.240710

In [28]:
```
X_train, X_test, y_train, y_test = train_test_split(df.drop('quality', axis=1),
                                                    df['quality'],
                                                    test_size=0.3,
                                                    random_state=0)


X_train.shape, X_test.shape
```

Out[28]:
```
((1119, 11), (480, 11))
```

In [29]:

```
# set up the scaler
scaler = StandardScaler()

# fit the scaler to the train set, it will learn the parameters
scaler.fit(X_train)

# transform train and test sets
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

In [30]:

```
dct = DecisionTreeClassifier(random_state = 42)
svc = SVC(random_state = 42)
rf = RandomForestClassifier(random_state = 42)
logreg = LogisticRegression(random_state = 42)
knn = KNeighborsClassifier()
sgd = SGDClassifier()
```

In [31]:

```
classifiers = [
    ("knn" , knn),
    ("rf" , rf),
    ("logreg" , logreg),
    ("svc", svc),
    ("sgd",sgd),
    ("dct",dct)
]
```

In [37]:

```
from sklearn.metrics import accuracy_score
```

In [38]:

```
for clf_name, clf in classifiers:
    # Fit clf to the training set
    clf.fit(X_train, y_train)

    # Predict y_pred
    y_pred = clf.predict(X_test)
    acc = accuracy_score(y_test, y_pred)

    # Evaluate clf's accuracy on the test set
    print('{:s} score : {:.3f}'.format(clf_name, acc))
```

```
knn score : 0.890
rf score : 0.921
logreg score : 0.906
svc score : 0.912
sgd score : 0.838
dct score : 0.892
```

In [39]:

```
param_grid ={
    'n_estimators' : [50,100,200],
    'max_depth': (1,5,10),
    'min_samples_leaf': (1,5,10)
    }
```

In [40]:
```
gridsearch = GridSearchCV(rf, param_grid=param_grid, scoring='accuracy', cv=5,n_jobs=6
```

In [41]:
```
%%capture
gridsearch.fit(X_train,y_train)
```

In [42]:
```
gridsearch.best_params_
```

Out[42]:
```
{'max_depth': 10, 'min_samples_leaf': 1, 'n_estimators': 100}
```

In [44]:
```
rf = RandomForestClassifier(max_depth= 10, min_samples_leaf = 1, n_estimators = 100)
```

In [45]:
```
rf.fit(X_train,y_train)
```

Out[45]:
```
RandomForestClassifier(max_depth=10)
```

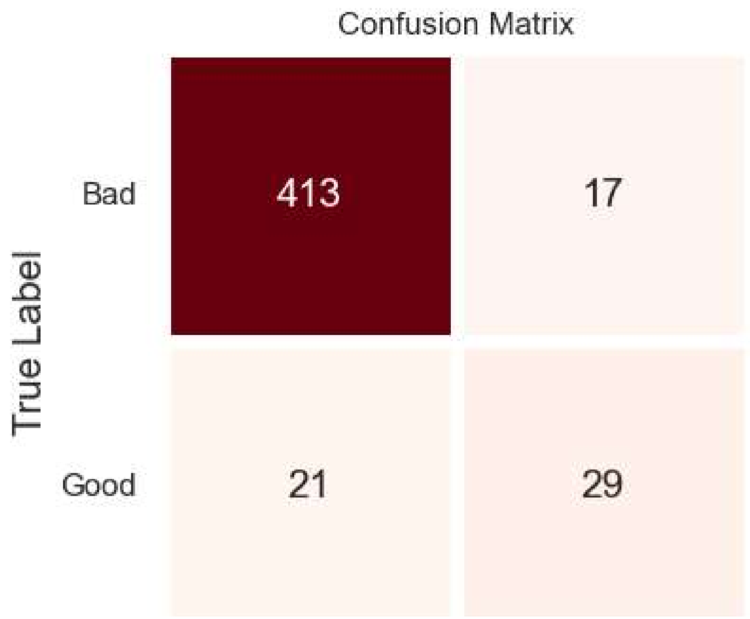
In [46]:
```
pred = rf.predict(X_test)
```

In [47]:
```
print("Accuracy Score:",accuracy_score(pred,y_test))
print("classification Report:\n",classification_report(pred,y_test))
```

```
Accuracy Score: 0.9208333333333333
classification Report:
               precision    recall  f1-score   support

           0       0.96      0.95      0.96       434
           1       0.58      0.63      0.60        46

    accuracy                           0.92       480
   macro avg       0.77      0.79      0.78       480
weighted avg       0.92      0.92      0.92       480
```

In [48]:

```
cm = confusion_matrix(y_test, pred)

df1 = pd.DataFrame(columns=["Bad","Good"], index= ["Bad","Good"], data= cm )

f,ax = plt.subplots(figsize=(4,4))

sns.heatmap(df1, annot=True,cmap="Reds", fmt= '.0f',ax=ax,linewidths = 5, cbar = False
plt.xlabel("Predicted Label")
plt.xticks(size = 12)
plt.yticks(size = 12, rotation = 0)
plt.ylabel("True Label")
plt.title("Confusion Matrix", size = 12)
plt.show()
```



In [ ]: