# Data Analytics Lab: Assignment - 5
# A mathematical essay on Random Forest

Nagappan N
*Department of Metallurgical and Materials Engineering*
*IIT Madras*
Chennai, India
mm19b040@smail.iitm.ac.in

*Abstract*—**The aim of this project is to predict the safety of a car based on its features. We use the random forest model to attain our goal. Our model makes this prediction with an accuracy of 96%.**

## I. Introduction

The main aim of our project is to predict the safety of a car depending on different features related to a car. Knowing the safety of a car is very crucial as thousands of people die everyday due to road accidents. Hence, the safety aspects of a car are important to save lives of people.

The Random Forest or Random Decision Forest is a supervised ensemble machine learning technique used for classification, regression, and other decision tree tasks. It is a form of bagging algorithm which works by creating multiple different decision trees.

In this project, we classify the cars based on its safety using the decision tree model. Our model is able to make this prediction at an accuracy of 96%. We use input features such as the price of the car, maintenance cost, features of the car and estimated safety for making our predictions.

This paper starts with the description of the random forest algorithm. It is followed by the description of the datasets. This includes data visualization. The following section includes details about data processing and how the model has been implemented. Finally, we conclude with the key inferences from our project.

## II. Random Forest

The Random forest or Random Decision Forest is a supervised ensemble Machine learning algorithm used for classification, regression, and other decision trees tasks. Random forest is a bagging algorithm. By creating additional data for training from a dataset by combining repeats and combinations to create multiple sets of the original data, the bagging strategy lowers prediction variance. The Random forest classifier builds a collection of decision trees from a randomly chosen portion of the training set. It is merely a collection of decision trees drawn from a randomly chosen subset of the training set, and it then gathers the votes from various decision trees to get the final prediction, using averaging to increase predictive accuracy and prevent over-fitting.

A decision tree is a type of tree structure that resembles a flowchart, where each internal node represents a test on an attribute, each branch a test result, and each leaf node (terminal node) a class label. The most important hyperparameters for decision tree are:

- Criterion: The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "log_loss" and "entropy.
- Max_depth: The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
- Min_samples_split: The minimum number of samples required to split an internal node.
- Min_samples_leaf: The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches.
- Min_weight_fraction_leaf: The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Samples have equal weight when sample_weight is not provided.
- Max_features: The number of features to consider when looking for the best split.
- Class_weight: Weights associated with classes in the form class_label: weight. If None, all classes are supposed to have weight one.

The sub-sample size is controlled with the max_samples parameter if bootstrap=True (default), otherwise the whole dataset is used to build each tree.

- Bootstrap: Whether bootstrap samples are used when building trees. If False, the whole dataset is used to build each tree.
- Max_samples: If bootstrap is True, the number of samples to draw from X to train each base estimator.

## III. Datasets

First we look at the features in our initial dataset. It has the following features:

1) buying: It is the buying price of the car. This is a categorical variable that takes one of the 4 values: vhigh, high, med, low.
2) maint: Price of maintenance which agains takes either vhigh, high, med or low as its value.
3) doors: Number of doors present in the car which can be 2, 3, 4, 5 or more.
4) persons: It is the number of people who can travel in the car which is also a categorical variable. It takes one of the follwing: 2, 4 or more.
5) lug_boot: This is a categorical variable that describes the size of the luggage boot. It can be small, med or big.
6) safety: This tells us about the estimated safety of the car. This feature takes the values low, med or high.
7) target: This is our target variable which again is a categorical variable that can take one of these values: unacc, acc, good, vgood.

## A. Data Visualization

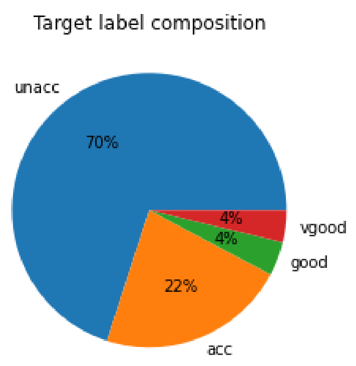


Fig. 1. Pie chart showing the proportion of each class in the target column.

We see that the proportion of classes is very unbalanced in the target column. A majority of them are unacc and only about 8% of them are good or vgood cumulatively.

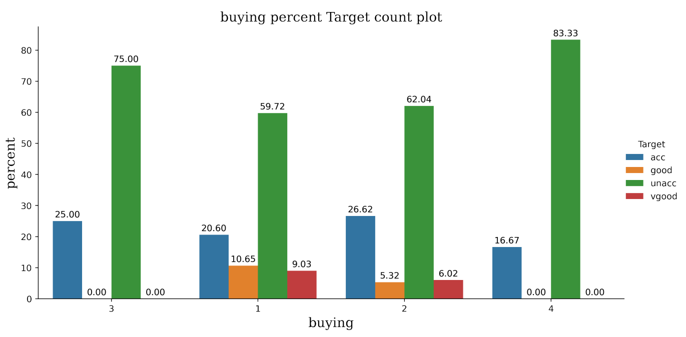


Fig. 2. Percentage of each target class based on the buying price of the car.

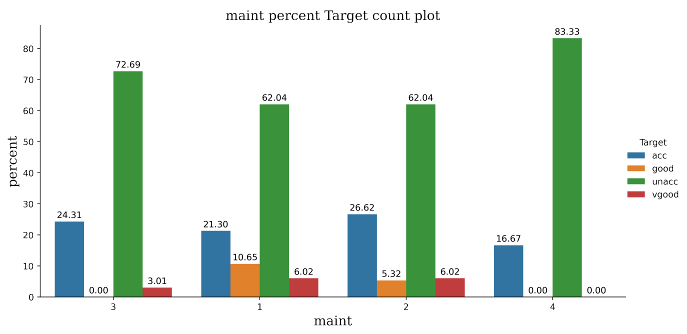


Fig. 3. Percentage of each target class based on the price of maintenance.

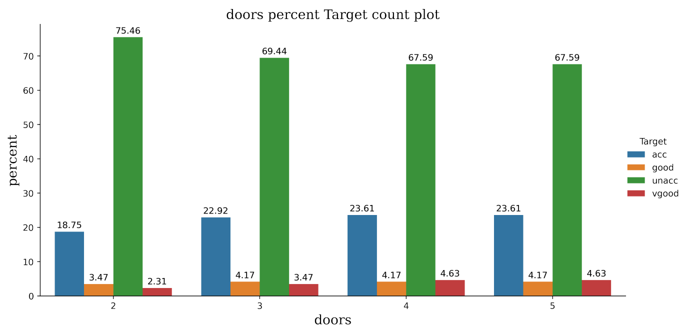


Fig. 4. Percentage of each target class based on the number of doors present in the car.

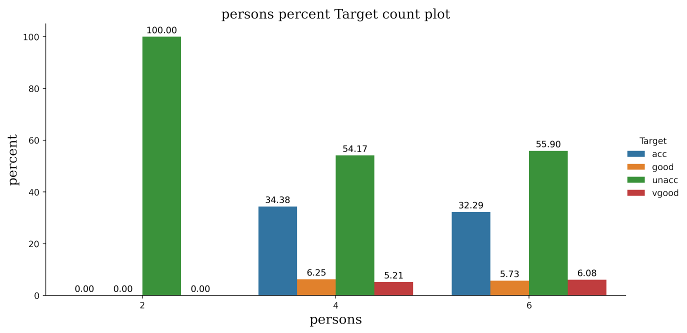


Fig. 5. Percentage of each target class based on the people capacity of the car

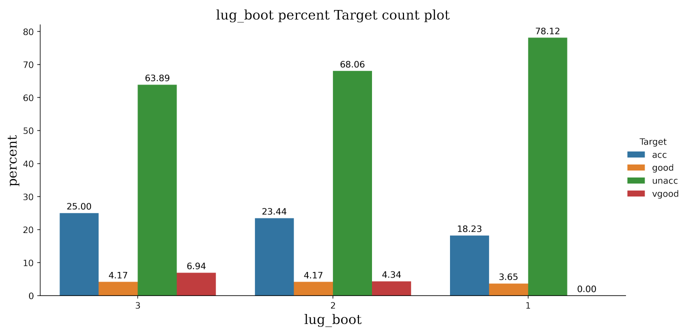


Fig. 6. Percentage of each target class based on the size of the luggage boot.

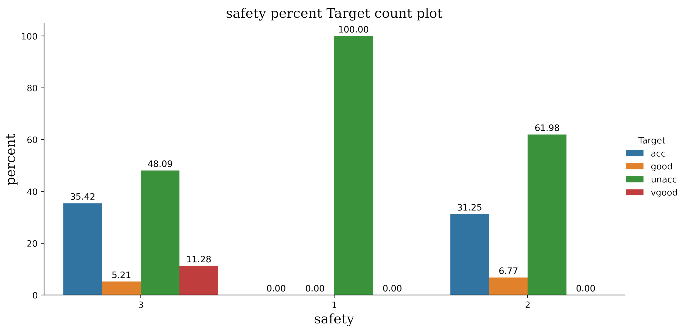


Fig. 7. Percentage of each target class based on the estimated safety of the car.

## IV. Modelling

All of the input features may be label encoded with ease because they are all ordinal in nature. Each one starts with one. Because the random forest module in Scikit Learn does not support nominal categorical variables, this is crucial to our model. Next, the train test split is carried out.

Prior to inputting data into our random forest model, we compare the composition of the target variable between the train and original compositions. We find that the target variables' composition hasn't altered notably due to the quantity of the dataset.

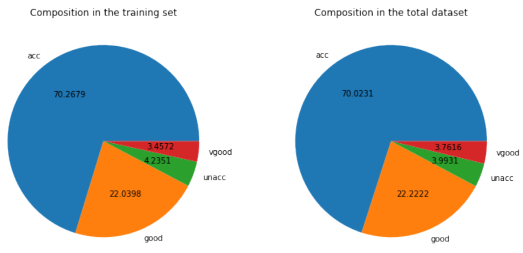
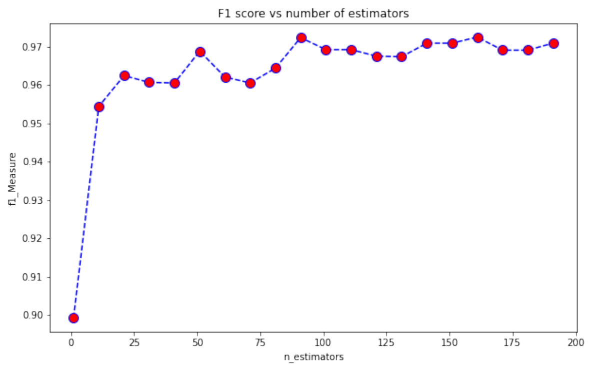
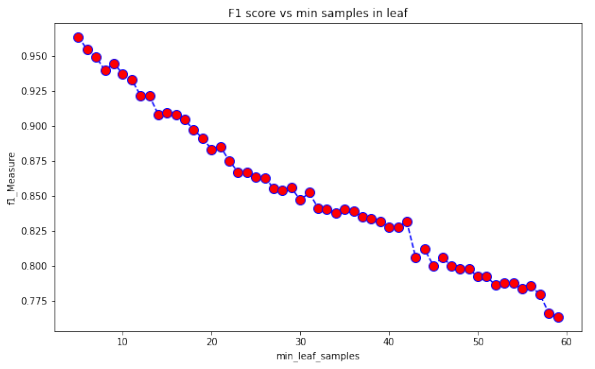


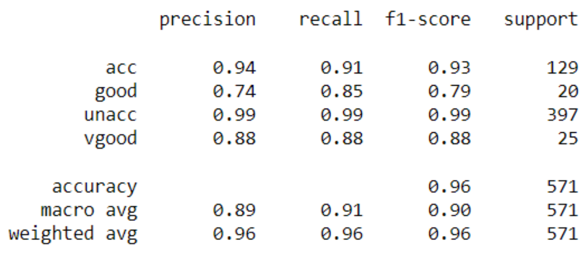
Fig. 8. Pie charts representing the composition of classes in the training dataset and the full dataset.

To determine the stability and maximum accuracy we can get from the model, we must first decide on the number of estimators for our random forest model. As a result, we have plotted the f1 weighted score against the number of estimators. When a n estimators value is selected above 100, we discover that we always obtain a stable and reasonable f1 score.



After selecting a n estimator of 160, we then decide on a min sample split for each decision tree in this model, and we discover that an optimum min samples leaf for each estimator would be 5 to avoid overfitting.





```
              precision    recall  f1-score   support

         acc       0.94      0.91      0.93       129
        good       0.74      0.85      0.79        20
       unacc       0.99      0.99      0.99       397
       vgood       0.88      0.88      0.88        25

    accuracy                           0.96       571
   macro avg       0.89      0.91      0.90       571
weighted avg       0.96      0.96      0.96       571
```

Fig. 11. Classification report

## V. Conclusions

We find that a model like a random forest is good to make predictions for real world problems like this. We have implemented a random forest model on the dataset to predict the safety of a car with an accuracy of 96%. We also see that the F1 scores for all the classes are good enough and thus our model is a good fit even for the imbalanced dataset.

# EE4708: Data Analytics Lab

## Assignment 5

**Name: Nagappan N**

**Roll number: MM19B040**

## Importing Libraries ¶

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

In [2]:
```
d = pd.read_excel('Data.xlsx')
```

In [3]:
```
d.head()
```

Out[3]:

| | buying | maint | doors | persons | lug_boot | safety | Target |
|---|---|---|---|---|---|---|---|
| **0** | vhigh | vhigh | 2 | 2 | small | low | unacc |
| **1** | vhigh | vhigh | 2 | 2 | small | med | unacc |
| **2** | vhigh | vhigh | 2 | 2 | small | high | unacc |
| **3** | vhigh | vhigh | 2 | 2 | med | low | unacc |
| **4** | vhigh | vhigh | 2 | 2 | med | med | unacc |

In [4]:
```
d.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1728 entries, 0 to 1727
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   buying    1728 non-null   object
 1   maint     1728 non-null   object
 2   doors     1728 non-null   object
 3   persons   1728 non-null   object
 4   lug_boot  1728 non-null   object
 5   safety    1728 non-null   object
 6   Target    1728 non-null   object
dtypes: object(7)
memory usage: 94.6+ KB
```

## Exploratory Data analysis of features

In [5]:
```
d['buying'].value_counts()
```

Out[5]:
```
vhigh    432
high     432
med      432
low      432
Name: buying, dtype: int64
```

In [6]:
```
d['maint'].value_counts()
```

Out[6]:
```
vhigh    432
high     432
med      432
low      432
Name: maint, dtype: int64
```

In [7]:
```
d['doors'].value_counts()
```

Out[7]:
```
2        432
3        432
4        432
5more    432
Name: doors, dtype: int64
```

In [8]:
```
d['persons'].value_counts()
```

Out[8]:
```
2       576
4       576
more    576
Name: persons, dtype: int64
```

In [9]:
```
d['lug_boot'].value_counts()
```

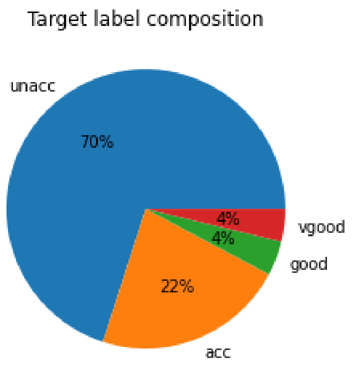
Out[9]:
```
small    576
med      576
big      576
Name: lug_boot, dtype: int64
```

In [10]:
```
d['safety'].value_counts()
```

Out[10]:
```
low     576
med     576
high    576
Name: safety, dtype: int64
```

We can see all the input features are well balanced

In [11]:
```
data=d['Target'].value_counts()
keys=d['Target'].value_counts().keys()
plt.title('Target label composition')
a=plt.pie(data, labels=keys, autopct='%.0f%%',)
plt.savefig('Targetpie.png')
```



The output target variable is an imbalanced multiclass variable as we can see from the pie chart above

In [12]:
```
d['buying'] = d['buying'].astype('category')
d['maint'] = d['maint'].astype('category')
d['doors'] = d['doors'].astype('category')
d['persons'] = d['persons'].astype('category')
d['lug_boot'] = d['lug_boot'].astype('category')
d['safety'] = d['safety'].astype('category')
d['Target'] = d['Target'].astype('category')
```

In [13]:
```
d.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1728 entries, 0 to 1727
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype   
---  ------    --------------  -----   
 0   buying    1728 non-null   category
 1   maint     1728 non-null   category
 2   doors     1728 non-null   category
 3   persons   1728 non-null   category
 4   lug_boot  1728 non-null   category
 5   safety    1728 non-null   category
 6   Target    1728 non-null   category
dtypes: category(7)
memory usage: 13.1 KB
```

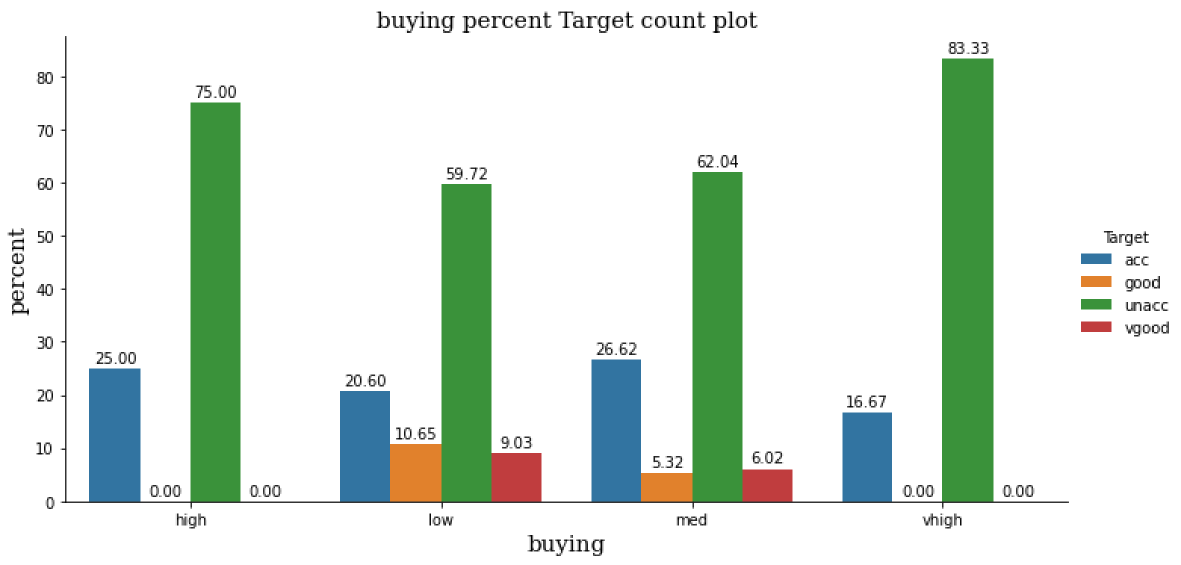
## Data Visualization

In [14]:
```
def funcplot(x1):
    y='Target'
    dfp=(d
    .groupby(x1)[y]
    .value_counts(normalize=True)
    .mul(100)
    .rename('percent')
    .reset_index())
    plt.figure(figsize=[15,7])
    g=sns.catplot(x=x1,y='percent',hue=y,data=dfp,kind='bar',height=5,aspect=2)
    for container in g.ax.containers:
        g.ax.bar_label(container, fmt='%.2f', padding=2)
    font1 = {'family':'serif','color':'black','size':15}
    font2 = {'family':'serif','color':'black','size':15}
    plt.title(x1+' percent Target count plot', fontdict=font1)
    plt.ylabel('percent',fontdict=font2)
    plt.xlabel(x1,fontdict=font2)
    plt.savefig(x1+'catplot.png',pad_inches=0,bbox_inches='tight',dpi=400)
```

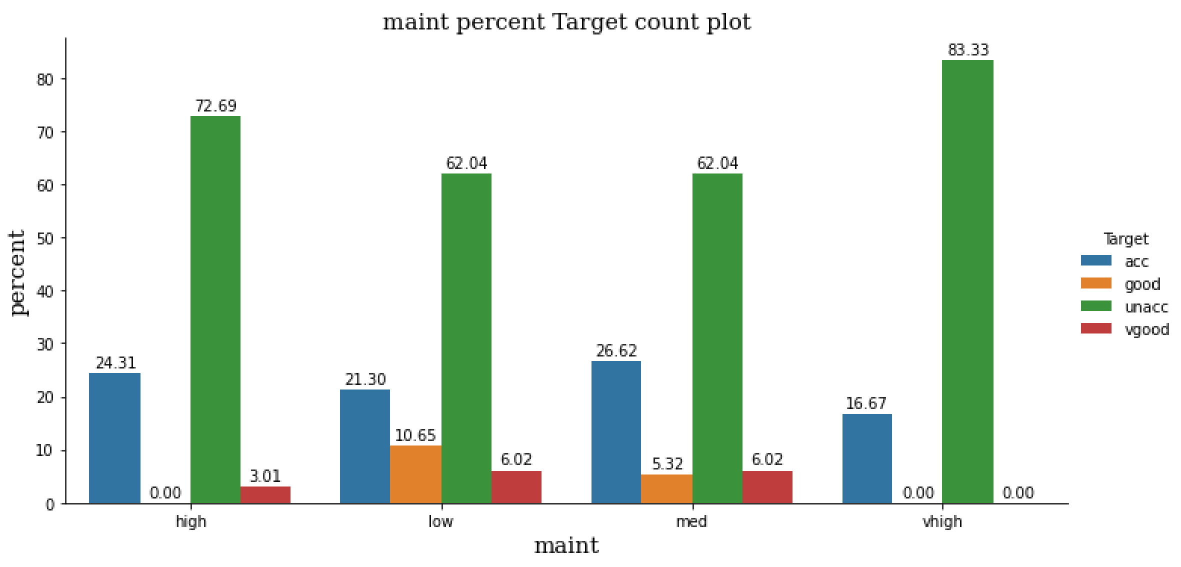
The following graphs visualize the composition of target class among each input variable category to analyse.

In [15]:
```
for x in d.columns:
    if(x!='Target'):
        funcplot(x)
```

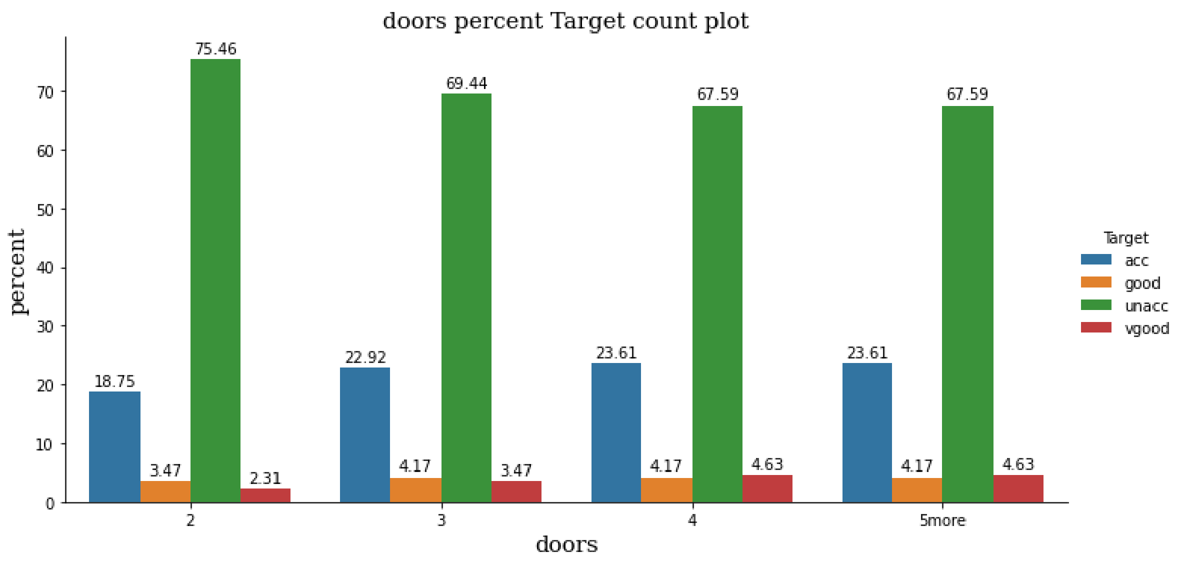
```
<Figure size 1080x504 with 0 Axes>
```



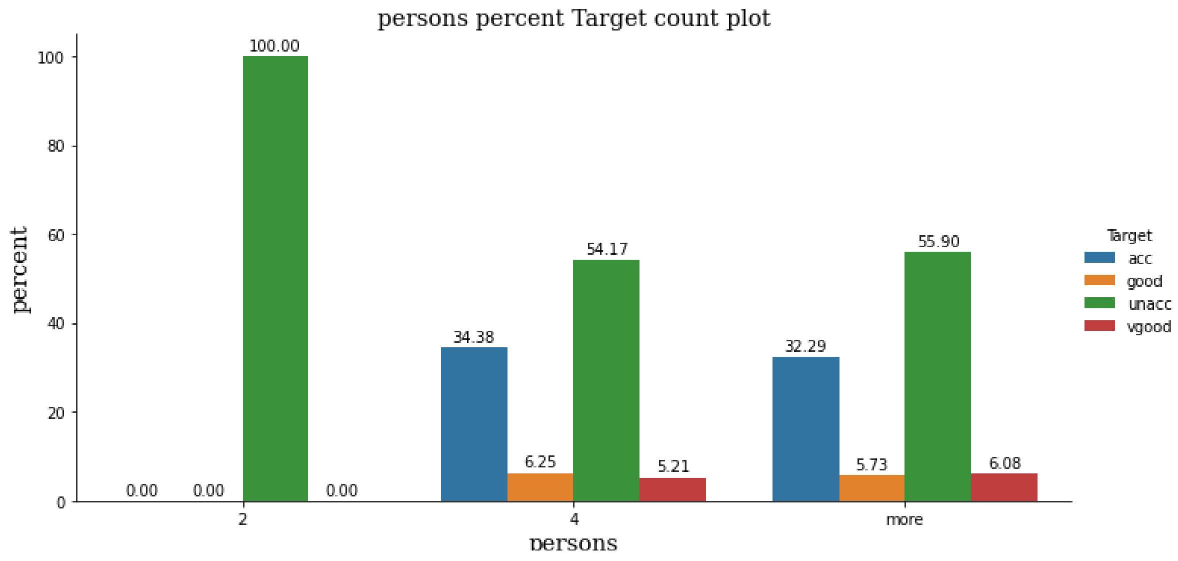
```
<Figure size 1080x504 with 0 Axes>
```



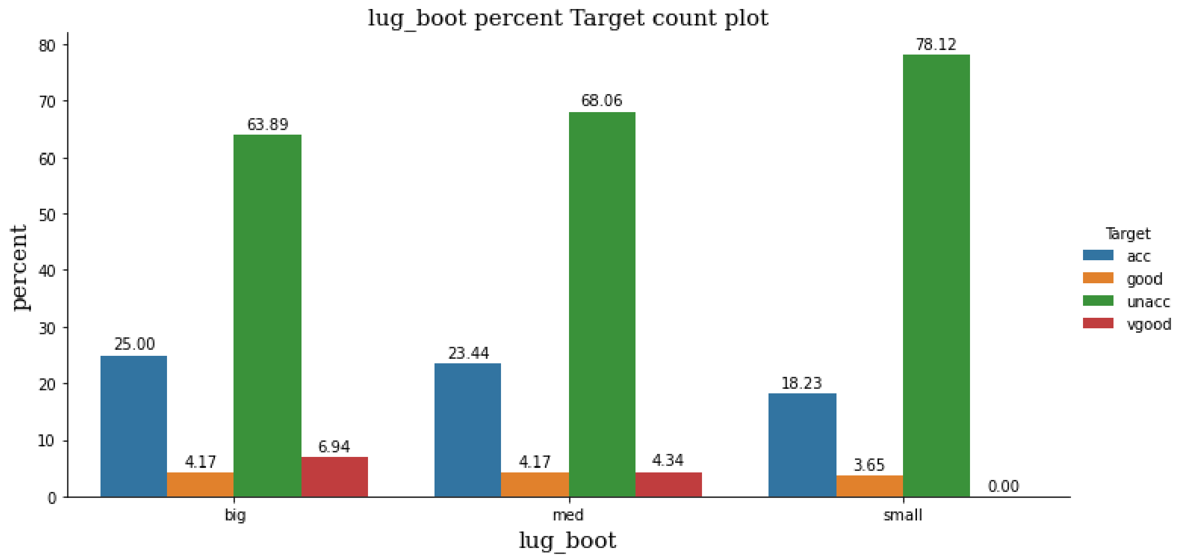
```
<Figure size 1080x504 with 0 Axes>
```



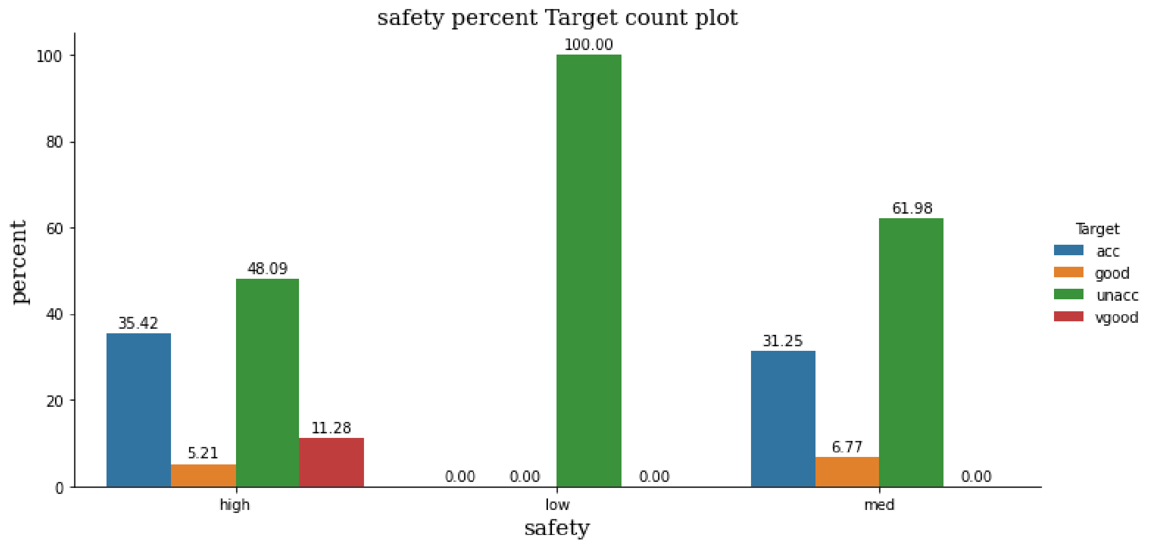
```
<Figure size 1080x504 with 0 Axes>
```


persons percent Target count plot
percent
persons
2
4
more
100.00
0.00
0.00
0.00
34.38
6.25
54.17
5.21
32.29
5.73
55.90
6.08
Target
acc
good
unacc
vgood

```
<Figure size 1080x504 with 0 Axes>
```


lug_boot percent Target count plot
percent
lug_boot
big
med
small
25.00
4.17
63.89
6.94
23.44
4.17
68.06
4.34
18.23
3.65
78.12
0.00
Target
acc
good
unacc
vgood

```
<Figure size 1080x504 with 0 Axes>
```


safety percent Target count plot
percent
safety
high
low
med
35.42
5.21
48.09
11.28
0.00
0.00
100.00
0.00
31.25
6.77
61.98
0.00
Target
acc
good
unacc
vgood

## Label Encoding of ordinal input features

As we observe all the input features are ordinal we just map them into respective labels.

In [16]:
```
mappings={'low':1,'med':2,'high':3,'vhigh':4,'5more':5,'more':6,'small':1,'big':3}
d.replace(mappings,inplace=True)
```

## Train Test Split

In [17]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report,f1_score
```

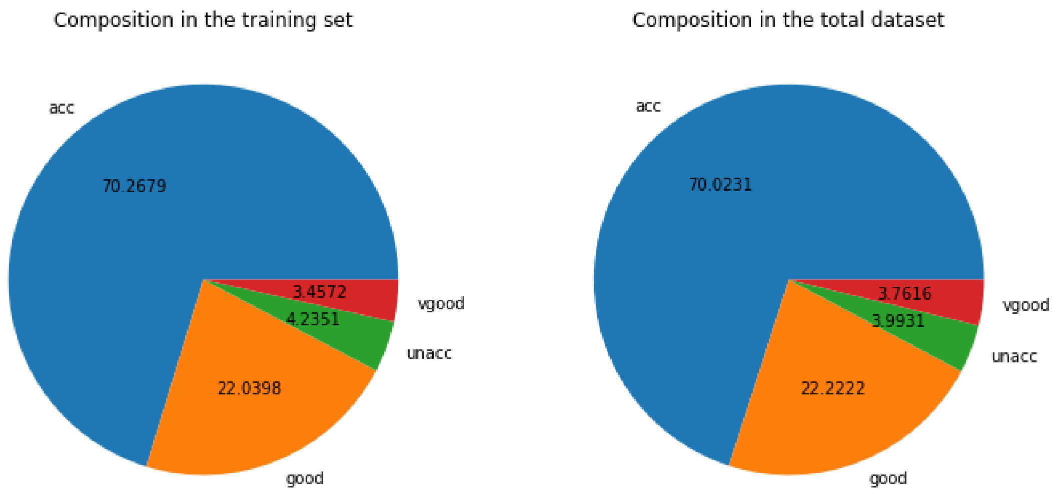
In [18]:
```
X = d.drop('Target',axis=1)
y = d['Target']
```

In [19]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

In [20]:
```
y_train.value_counts(normalize=True)
```

Out[20]:
```
unacc    0.702679
acc      0.220398
good     0.042351
vgood    0.034572
Name: Target, dtype: float64
```

In [21]:
```
plt.figure(figsize=(12,8))
plt.subplot(1,2,1)
data=y_train.value_counts(normalize=True)
keys=np.unique(y_train)
data2=y.value_counts(normalize=True)
plt.title('Composition in the training set')
a=plt.pie(data, labels=keys, autopct='%.4f')
plt.subplot(1,2,2)
plt.title('Composition in the total dataset')
b=plt.pie(data2, labels=keys, autopct='%.4f')
plt.savefig('compo.png')
```



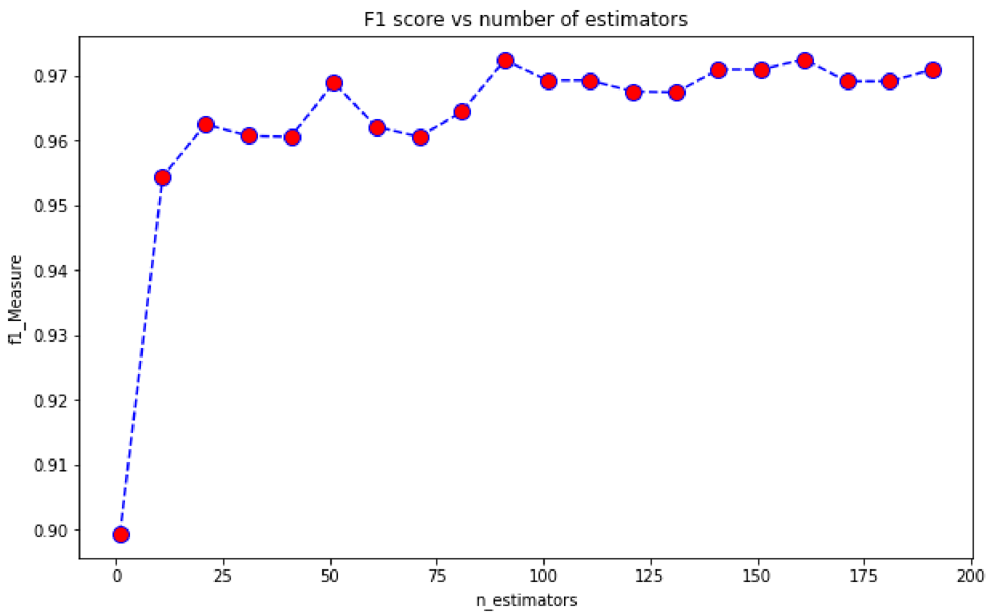
In [22]:
```
y.value_counts(normalize=True)
```

Out[22]:
```
unacc    0.700231
acc      0.222222
good     0.039931
vgood    0.037616
Name: Target, dtype: float64
```

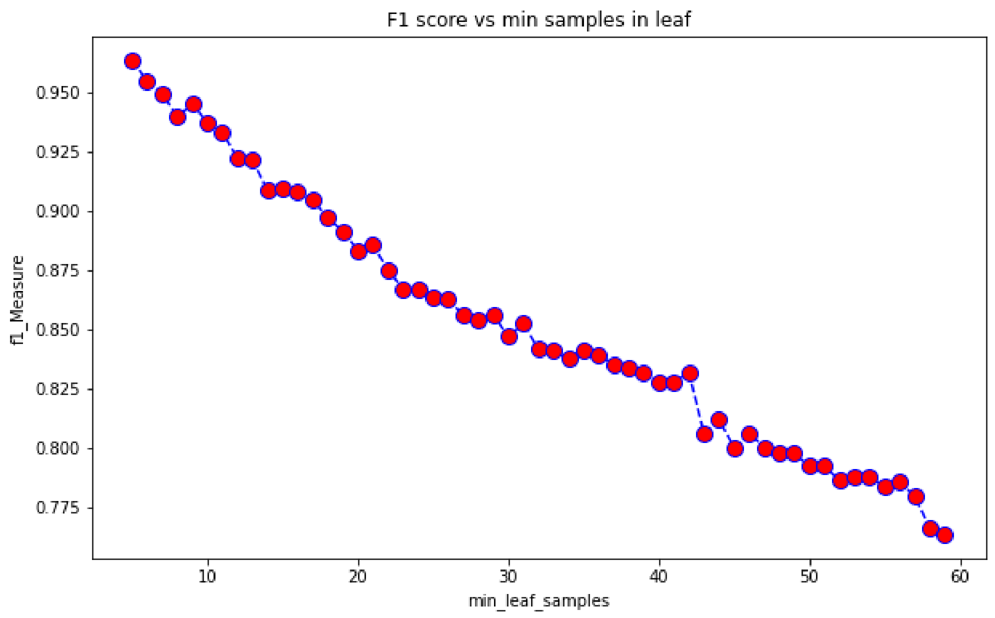
## Random Forest

In [23]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [32]:
```
a=[]
for i in range(1,200,10):
    rf = RandomForestClassifier(n_estimators=i,criterion='gini',random_state=101)
    rf.fit(X_train,y_train)
    pred=rf.predict(X_test)
    a.append(f1_score(y_test,pred,average='weighted'))
plt.figure(figsize=(10,6))
plt.plot(range(1,200,10),a,color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10)
plt.title('F1 score vs number of estimators')
plt.xlabel('n_estimators')
plt.ylabel('f1_Measure')
plt.savefig('f1_n.png')
```



In [33]:
```
b=[]
for i in range(5,60,1):
    rf = RandomForestClassifier(n_estimators=160,criterion='gini',random_state=101,min_samples_le
    rf.fit(X_train,y_train)
    pred=rf.predict(X_test)
    b.append(f1_score(y_test,pred,average='weighted'))
plt.figure(figsize=(10,6))
plt.plot(range(5,60,1),b,color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10)
plt.title('F1 score vs min samples in leaf')
plt.xlabel('min_leaf_samples')
plt.ylabel('f1_Measure')
plt.savefig('f1_m.png')
```



In [26]:
```
rf=RandomForestClassifier(n_estimators=160,random_state=101,min_samples_leaf=5)
rf.fit(X_train,y_train)
```

Out[26]:
```
RandomForestClassifier(min_samples_leaf=5, n_estimators=160, random_state=101)
```

In [27]:
```
pred= rf.predict(X_test)
```

In [28]:
```
confusion_matrix(y_test, pred)
```

Out[28]:
```
array([[118,   6,   4,   1],
       [  1,  17,   0,   2],
       [  4,   0, 393,   0],
       [  3,   0,   0,  22]], dtype=int64)
```

```
In [29]:  print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

         acc       0.94      0.91      0.93       129
        good       0.74      0.85      0.79        20
       unacc       0.99      0.99      0.99       397
       vgood       0.88      0.88      0.88        25

    accuracy                           0.96       571
   macro avg       0.89      0.91      0.90       571
weighted avg       0.96      0.96      0.96       571
```