Q1-car-data

March 8, 2021

0.0.1 Question 1

0.0.2 The following dataset is used to classify the car acceptability into classes: unacceptable, acceptable, good and very good.

1. Import the car.data dataset from https://archive.ics.uci.edu/ml/machine-learning-databases/car/ (Links to an external site.)
2. Extract X as all columns except the last column and Y as last column.
3. Visualize the dataset using any two appropriate graphs.
4. Visualize the correlation between all the variables of dataset.
5. Split the data into training set and testing set.Perform 10-fold cross validation
6. Train a Logistic regression model for the dataset.
7. Can we compute the accuracy and confusion matrix.
8. Predict and display the class label of a car with following attributes : buying, main, doors, persons, lug_boot, safety as [vhigh,low,4,more,small,med].

[1]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import make_classification
from sklearn.model_selection import learning_curve
import seaborn as sns
import sklearn as sk
import plotly.offline as py
from plotly import tools
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

0.1 1. Import the car.data dataset from https://archive.ics.uci.edu/ml/machine-learning-databases/car/ (Links to an external site.)

[2]:
```
# Import data files directly from provided URL
file_one = "https://archive.ics.uci.edu/ml/machine-learning-databases/car/car.
↪data"
```

```
file_two = "https://archive.ics.uci.edu/ml/machine-learning-databases/car/car.
 ↪names"
file_three = "https://archive.ics.uci.edu/ml/machine-learning-databases/car/car.
 ↪c45-names"

# Define Attribute Labels
attributes=['buying','maint','doors','persons','lug_boot','safety', 'class']

# Read Data files
car_data = pd.read_csv(file_one, names=attributes)
car_names = pd.read_csv(file_two, sep="\t")
car_c = pd.read_csv(file_three, sep="|")
car_data.head()
```

[2]:

	buying	maint	doors	persons	lug_boot	safety	class
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

1.1 Get the Summary of the Data set and check for Null and Unique values

[3]:

```
# Let's see a summary of our dataframe
print ("Shape of Data :" , car_data.shape)
print ("Features :" ,car_data.columns.tolist())
print ("Missing values : ", car_data.isnull().sum().values.sum())
print ("Unique values :  \n", car_data.nunique())
```

```
Shape of Data : (1728, 7)
Features : ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety',
'class']
Missing values :  0
Unique values :
 buying      4
maint       4
doors       4
persons     3
lug_boot    3
safety      3
class       4
dtype: int64
```

- This Data Set contains “No Missing Values”, 6 features and 1 Target Class and each Column has 0-4 unique values.

1.2 Data distribution among each column

[4]:
```python
for i in car_data.columns:
    print(car_data[i].value_counts())
    print()
```

```
vhigh    432
high     432
med      432
low      432
Name: buying, dtype: int64

vhigh    432
high     432
med      432
low      432
Name: maint, dtype: int64

4        432
5more    432
2        432
3        432
Name: doors, dtype: int64

4       576
2       576
more    576
Name: persons, dtype: int64

med      576
small    576
big      576
Name: lug_boot, dtype: int64

high    576
med     576
low     576
Name: safety, dtype: int64

unacc    1210
acc       384
good       69
vgood      65
Name: class, dtype: int64
```

- Data distribution depicts that all the columns are distributed equally except the Class. Class Column looks unabalanced as 70% of the values are 'Unacceptable'.

Let's explore the Class data.

[5]:
```
# Check unique values of Target attribute
print('Unique Values of Class feature: ',car_data['class'].unique())
print('Value Count :\n',car_data['class'].value_counts())
```

```
Unique Values of Class feature:  ['unacc' 'acc' 'vgood' 'good']
Value Count :
 unacc    1210
acc       384
good       69
vgood      65
Name: class, dtype: int64
```

0.2 2. Extract X as all columns except the last column and Y as last column.

- Classify the data into input and output columns, X and Y colums, features and class variables. ##### This Classification creates the dataset and confirms the expected number of samples and features.

[6]:
```
# Separate into input and output columns
X = car_data.iloc[:, :6]
y = car_data.iloc[:,-1]

# Summarize the data set
print('Shape of the data set :',X.shape, y.shape)
print('X:\n',X.head())
print('y:\n',y.head())
```

```
Shape of the data set : (1728, 6) (1728,)
X:
   buying  maint doors persons lug_boot safety
0  vhigh  vhigh     2       2    small    low
1  vhigh  vhigh     2       2    small    med
2  vhigh  vhigh     2       2    small   high
3  vhigh  vhigh     2       2      med    low
4  vhigh  vhigh     2       2      med    med
y:
 0    unacc
1    unacc
2    unacc
3    unacc
4    unacc
Name: class, dtype: object
```

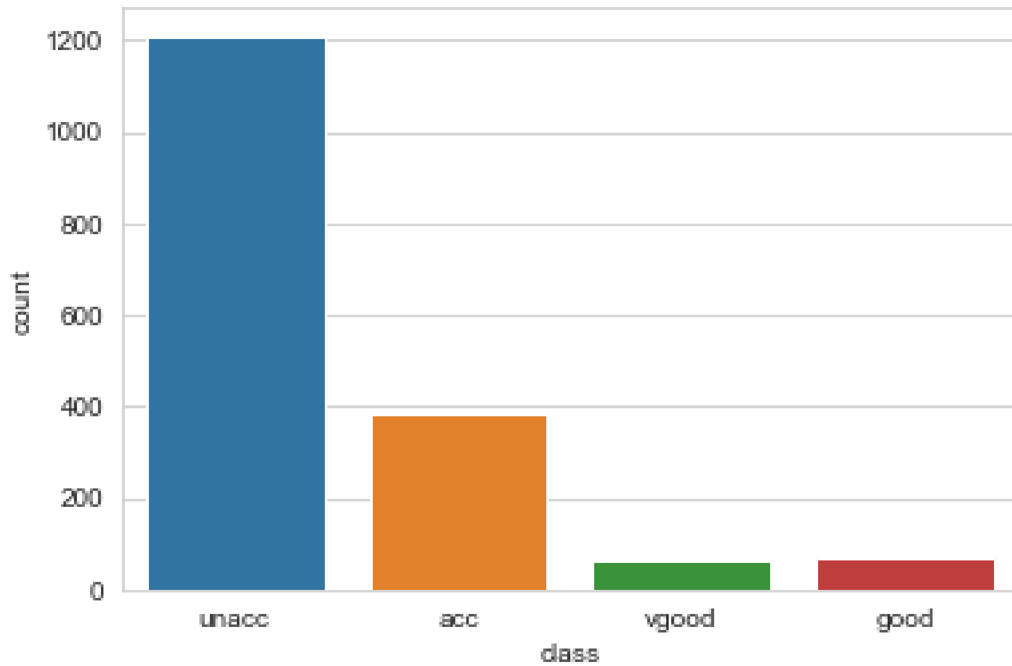
0.3 3. Visualize the dataset using any two appropriate graphs

3.1 Plot Class Value Count Values

[7]:
```
# Plot Class Value Count Values
sns.set_style('whitegrid')
```

```
sns.countplot(x='class', data=car_data)
```

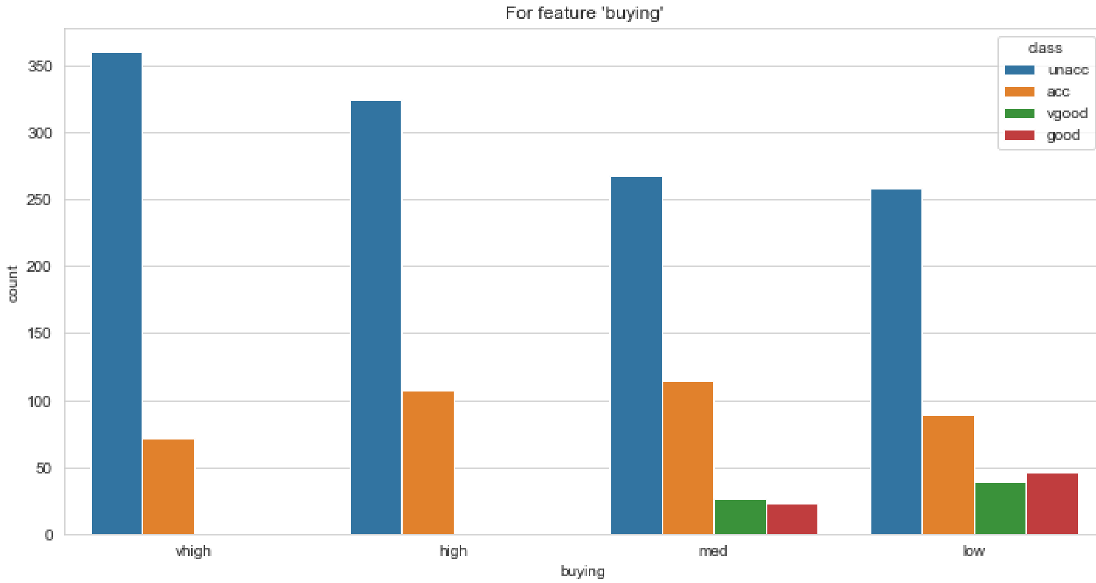
[7]: <matplotlib.axes._subplots.AxesSubplot at 0x170a5f2c0b8>

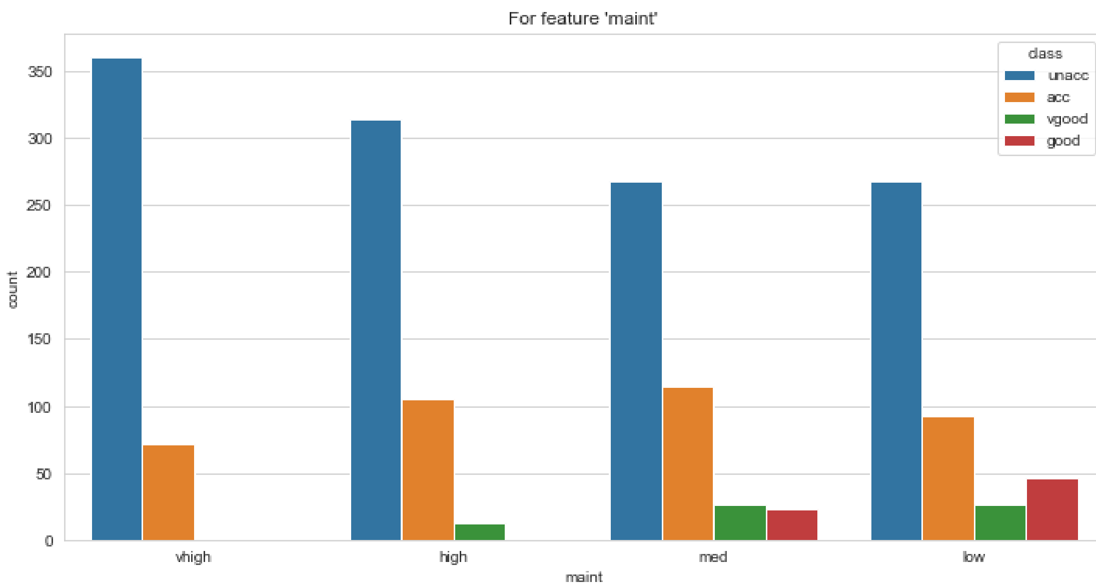


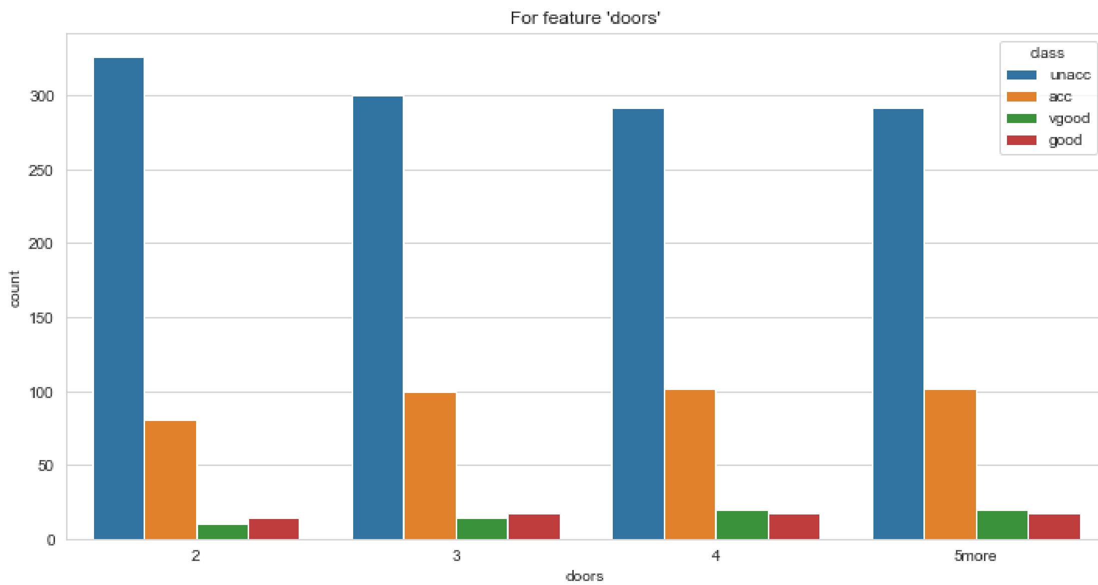
- Visual 1 - The Bar graph of the distribution of class values clearly shows that data is imbalanced as Class Data has large number of "unacc" values. This is a typical example of **unbalance classification problem

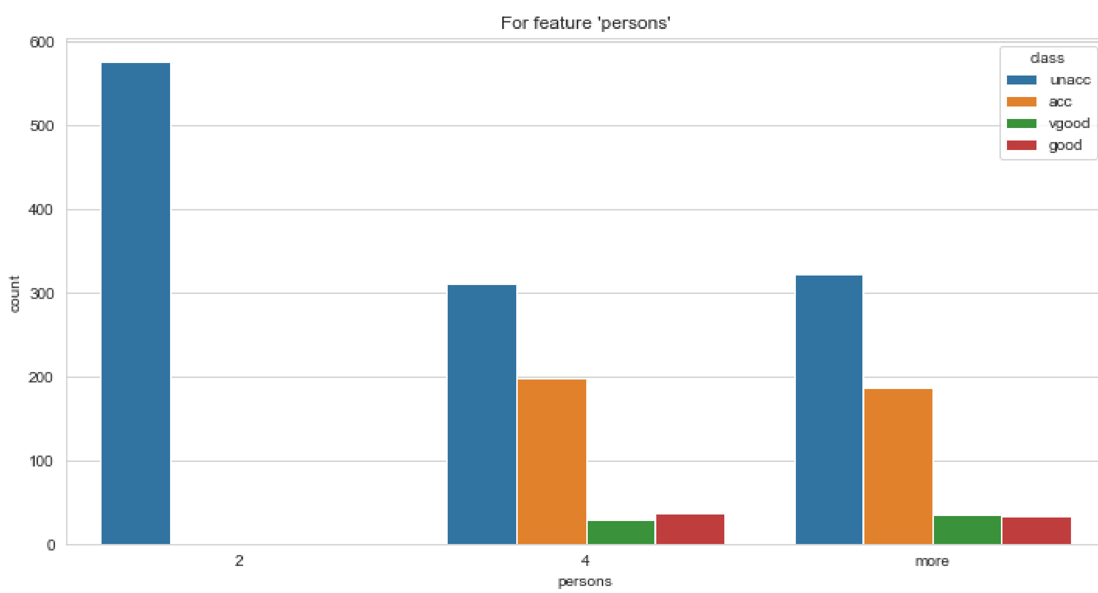
3.2 Plot all Feature across class value

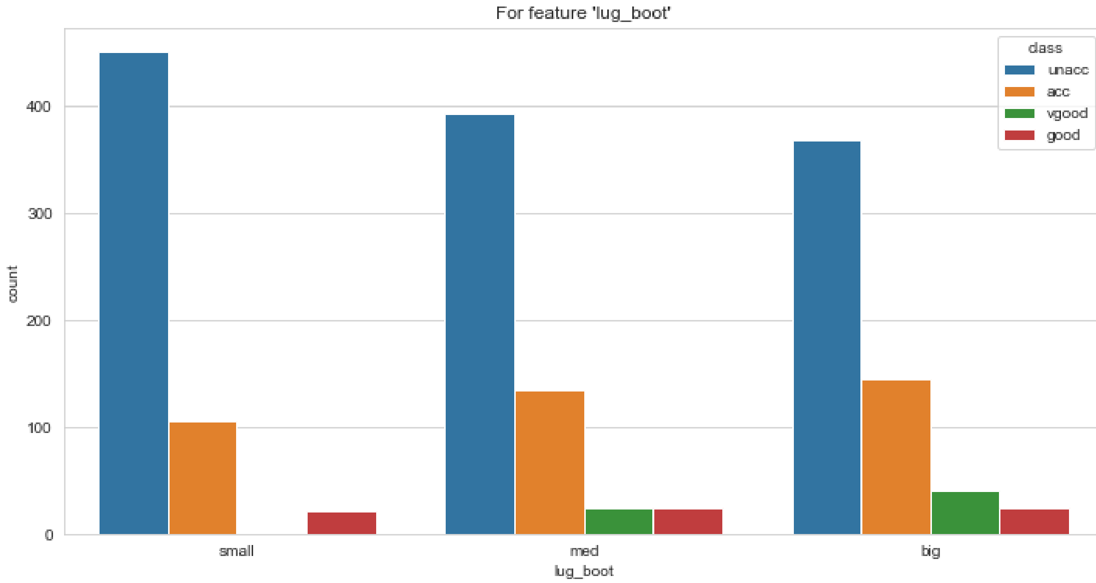
[8]:
```
# Summarize All features across class value
for i in car_data.columns[:-1]:
    plt.figure(figsize=(12,6))
    plt.title("For feature '%s'" %i)
    sns.countplot(car_data[i],hue=car_data['class'])
```

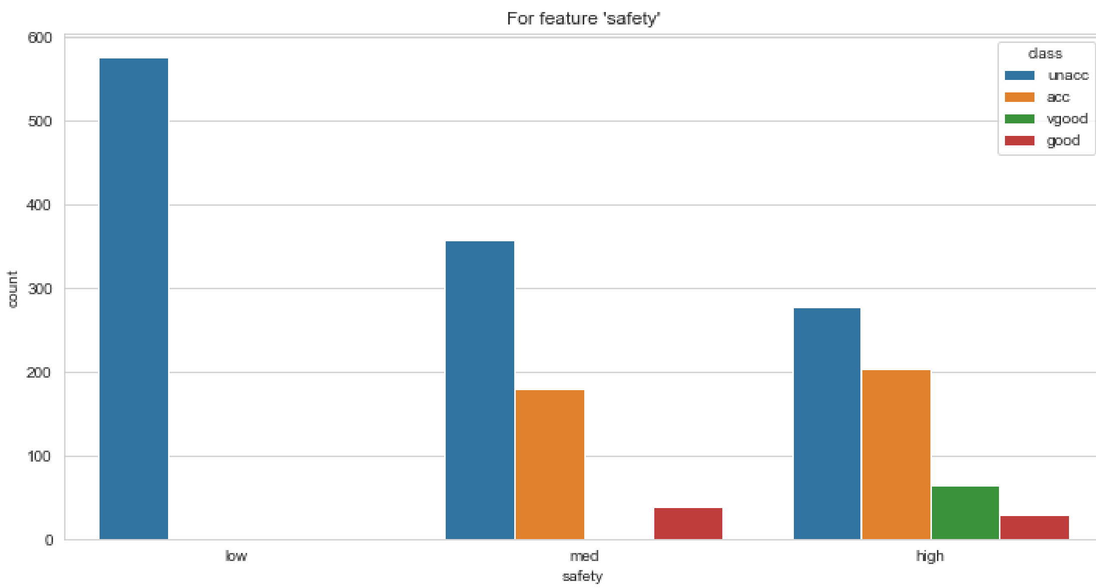

For feature 'buying'
class
unacc
acc
vgood
good
count
0
50
100
150
200
250
300
350
vhigh
high
med
low
buying


For feature 'maint'
class
unacc
acc
vgood
good
count
0
50
100
150
200
250
300
350
vhigh
high
med
low
maint


For feature 'doors'
class
unacc
acc
vgood
good
count
0
50
100
150
200
250
300
2
3
4
5more
doors


For feature 'persons'
class
unacc
acc
vgood
good
count
0
100
200
300
400
500
600
2
4
more
persons





Visual 2 - The Bar graph of the distribution of all features across the class value. Looking at the visual we can say that data is distributed evenly.

3.3 Check the percentage of the class distrbution Install plotly to view this graph(pip install plotly)

[9]:
```
fig = {
  "data": [
```

```
    {
      "values": [1210,384,69,65],
      "labels": [
        "Unacceptable",
        "Acceptable",
        "Good",
        "Very Good"
      ],
      "domain": {"column": 0},
      "name": "Car Quality",
      "hoverinfo":"label+percent+name",
      "hole": .6,
      "type": "pie"
    }],
  "layout": {
        "title":"Distribution of Cars",
        "grid": {"rows": 1, "columns": 1},
        "annotations": [
            {
                "font": {
                    "size": 36
                },
                "showarrow": False,
                "text": "",
                "x": 0.5,
                "y": 0.5
            }
        ]
    }
}
py.iplot(fig, filename='plot')
```

- Visual 3: We can see from the graph, 70% of data is of "unacc" class.

0.4 4. Visualize the correlation between all the variables of dataset

To Perform Correlation and to Apply Machine Learning algorithm, convert all the Categorical values to numerical values

4.1 Factorizing Categorical data to Numerical data

[10]:
```
#Convert Object to Integer Values
# X Column
car_data['buying'],buying = pd.factorize(car_data['buying'])
car_data['maint'],maint = pd.factorize(car_data['maint'])
car_data['doors'],doors = pd.factorize(car_data['doors'])
car_data['persons'],persons = pd.factorize(car_data['persons'])
car_data['lug_boot'],lug_boot = pd.factorize(car_data['lug_boot'])
```

```
car_data['safety'],safety = pd.factorize(car_data['safety'])
# Y column
car_data['class'],class_names = pd.factorize(car_data['class'])

print(car_data.head())
```

```
   buying  maint  doors  persons  lug_boot  safety  class
0       0      0      0        0         0       0      0
1       0      0      0        0         0       1      0
2       0      0      0        0         0       2      0
3       0      0      0        0         1       0      0
4       0      0      0        0         1       1      0
```

[11]:
```
print(car_data['buying'].unique())
print("buying {}".format(buying))
print(car_data['maint'].unique())
print("Maintanance {}".format(maint))
print(car_data['doors'].unique())
print("doors {}".format(doors))
print(car_data['persons'].unique())
print("persons {}".format(persons))
print(car_data['lug_boot'].unique())
print("lug_boot {}".format(lug_boot))
print(car_data['safety'].unique())
print("safety {}".format(safety))
print(car_data['class'].unique())
print("class {}".format(class_names))
```

```
[0 1 2 3]
buying Index(['vhigh', 'high', 'med', 'low'], dtype='object')
[0 1 2 3]
Maintanance Index(['vhigh', 'high', 'med', 'low'], dtype='object')
[0 1 2 3]
doors Index(['2', '3', '4', '5more'], dtype='object')
[0 1 2]
persons Index(['2', '4', 'more'], dtype='object')
[0 1 2]
lug_boot Index(['small', 'med', 'big'], dtype='object')
[0 1 2]
safety Index(['low', 'med', 'high'], dtype='object')
[0 1 2 3]
class Index(['unacc', 'acc', 'vgood', 'good'], dtype='object')
```

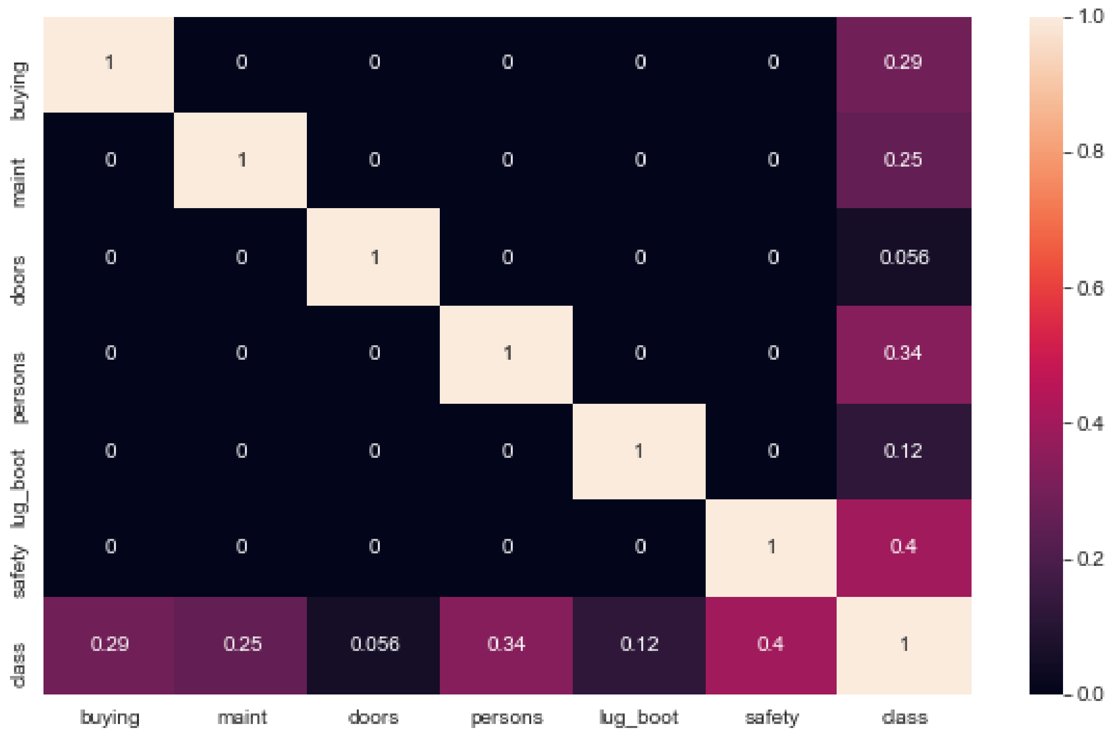
Extract X and Y columns after converting the data from categorical to Numerical

[12]:
```
X = car_data.iloc[:, :6]
y = car_data.iloc[:,-1]
```

4.2 Plot Correlation between all variables in the data set

[13]:
```
#  Plotting Correlation of features
plt.figure(figsize=(10,6))
sns.heatmap(car_data.corr(), annot=True)
```

[13]: <matplotlib.axes._subplots.AxesSubplot at 0x170a8c94668>



- Visual 4 : The Correlation heatmap shows, there is no correlation among all these features. #### Except for the class, it has a very weak correlation with other features. "doors" and "lug_boot" columns has very weak correlation with "class".

0.5 5. Split the data into training set and testing set.Perform 10-fold cross validation

5.1 Split the dataset into train and test sets

[14]:
```
from sklearn.model_selection import train_test_split
# split data randomly into 80% training and 20% test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,␣
↪random_state=9)
```

5.2 Apply 10-Fold Cross Validation

[15]:
```
from numpy import mean
from numpy import std
```

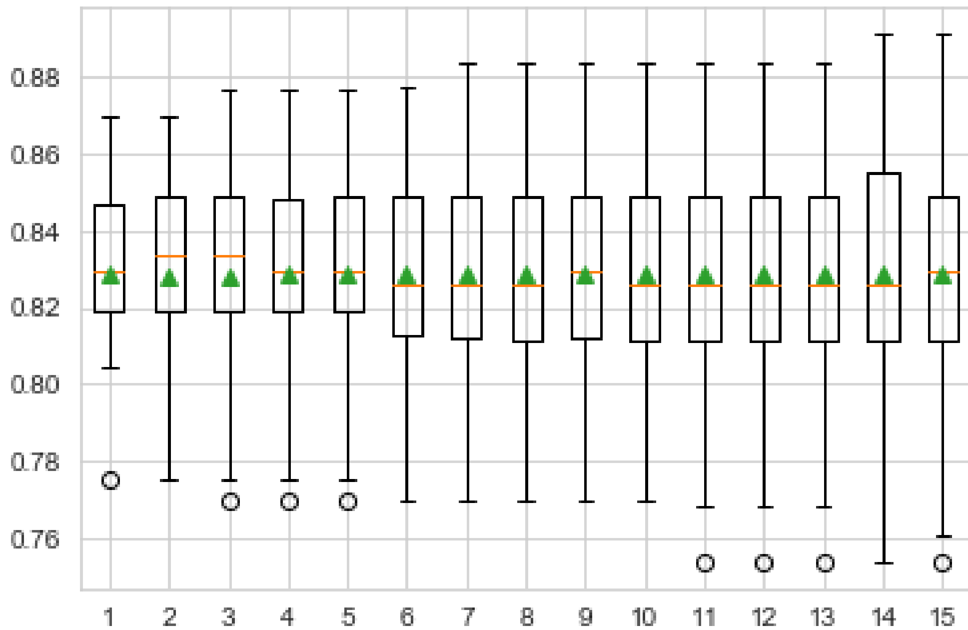
```
from sklearn.datasets import make_classification
from sklearn.model_selection import KFold, RepeatedKFold
from sklearn.model_selection import cross_val_score
from scipy.stats import sem
from numpy import mean
from numpy import std
```

5.2.1 Evaluating the mean and standard error classification accuracy using 10-fold cross-validation with 15 numbers of repeats.

[16]:
```
# evaluate a model with a given number of repeats
def evaluate_model(X, y, repeats):
    # prepare the cross-validation procedure
    cv = RepeatedKFold(n_splits=10, n_repeats=repeats, random_state=1)
    # create model
    model = LogisticRegression()
    # evaluate model
    scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
    return scores

X = X_train
y = y_train
# configurations to test
repeats = range(1,16)
results = list()
for r in repeats:
    # evaluate using a given number of repeats
    scores = evaluate_model(X, y, r)
    # summarize
    print('>%d mean=%.4f se=%.3f' % (r, mean(scores), sem(scores)))
    # score
    results.append(scores)
# plot the results
plt.boxplot(results, labels=[str(r) for r in repeats], showmeans=True)
plt.show()
```

```
>1 mean=0.8285 se=0.008
>2 mean=0.8278 se=0.006
>3 mean=0.8283 se=0.005
>4 mean=0.8285 se=0.004
>5 mean=0.8288 se=0.004
>6 mean=0.8287 se=0.003
>7 mean=0.8288 se=0.003
>8 mean=0.8288 se=0.003
>9 mean=0.8289 se=0.003
>10 mean=0.8286 se=0.003
>11 mean=0.8288 se=0.003
>12 mean=0.8288 se=0.003
```

```
>13 mean=0.8288 se=0.003
>14 mean=0.8289 se=0.003
>15 mean=0.8289 se=0.002
```



- We can see that the mean seems to coalesce around a value of about 82.8 percent. We might take this as the stable estimate of model performance
- Looking at the standard error, we can see that it decreases with an increase in the number of repeats and stabilizes with a value around 0.003 at around 7 or 8 repeats.
- A box and whisker plot is created to summarize the distribution of scores for each number of repeats. The orange line indicates the median of the distribution and the green triangle represents the arithmetic mean.

0.6 6. Train a Logistic regression model for the dataset.

1. Since the Dataset contains more than two classes, it is a multiclass classification problem. so we have applied Multinomial Logistic regression.
2. The LogisticRegression class can be configured for multinomial logistic regression by setting the "multi_class" argument to "multinomial" and the "solver" argument to a solver that supports multinomial logistic regression, such as 'newton-cg', 'lbfgs', 'sag'.
3. we found the best paramter to get good accuracy by applying Hyperparameter tuning to our model using GridSearchCV

Apply Logistic Regression

[17]:
```
logmodel = LogisticRegression(solver='newton-cg', max_iter=1000,␣
 ↪multi_class='multinomial', C=10, random_state=6)
logmodel.fit(X_train,y_train)
```

```
predictions = logmodel.predict(X_test)
```

Hyperparameter tuning is applied to get good accuracy of the model.

[18]:
```
from sklearn.model_selection import GridSearchCV
param_grid={'C':[0.01,0.1,1,10],
            'solver':['newton-cg', 'lbfgs', 'sag'],
            'multi_class':['multinomial']}
grid=GridSearchCV(estimator=LogisticRegression(n_jobs=-1),param_grid=param_grid,cv=5,n_jobs=-1
grid.fit(X_train,y_train)
print('Grid Parameters : ',grid.best_params_)
print('Gris Score : ',grid.best_score_)
```

```
Grid Parameters :  {'C': 10, 'multi_class': 'multinomial', 'solver': 'newton-
cg'}
Gris Score :  0.8314000941767384
```

0.7 7. Computing the accuracy and confusion matrix.

[19]:
```
from sklearn.metrics import classification_report, accuracy_score,␣
↪confusion_matrix, f1_score, precision_score, recall_score

# Print Classification Report
print('Classification repor:t\n',classification_report(y_test,predictions))

# create Confusion Matrix
cm = confusion_matrix(y_test, predictions)
print('Confusion Matrix : \n',cm)

# Compute Accuracy
accuracy = accuracy_score(y_test, predictions)
print('\nAccuracy: {:.2f}'.format(accuracy))
```

```
Classification repor:t
               precision    recall  f1-score   support

           0       0.90      0.91      0.90       238
           1       0.68      0.67      0.68        79
           2       0.92      0.80      0.86        15
           3       0.60      0.64      0.62        14

    accuracy                           0.84       346
   macro avg       0.78      0.76      0.76       346
weighted avg       0.84      0.84      0.84       346

Confusion Matrix :
 [[216  17   0   5]
 [ 24  53   1   1]
```

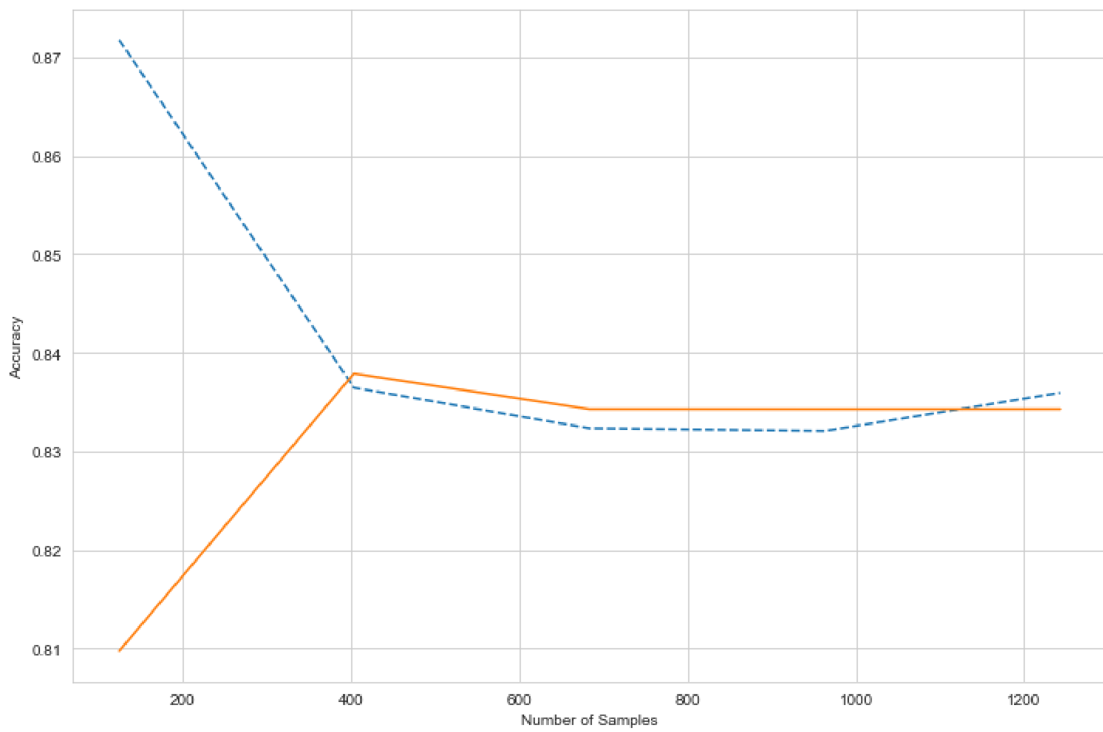
```
[  0   3  12   0]
[  0   5   0   9]]

Accuracy: 0.84
```

0.8 8. Plot the decision boundary, visualize training and test results.

8.1 Plot Learning curve for Logistic Regression Model (Training and Test)

[20]:
```
lc=learning_curve(logmodel,X_train,y_train,cv=10,n_jobs=-1)
size=lc[0]
train_score=[lc[1][i].mean() for i in range (0,5)]
test_score=[lc[2][i].mean() for i in range (0,5)]
fig=plt.figure(figsize=(12,8))
plt.xlabel("Number of Samples")
plt.ylabel("Accuracy")
plt.plot(size,train_score, ls='--')
plt.plot(size,test_score)
```

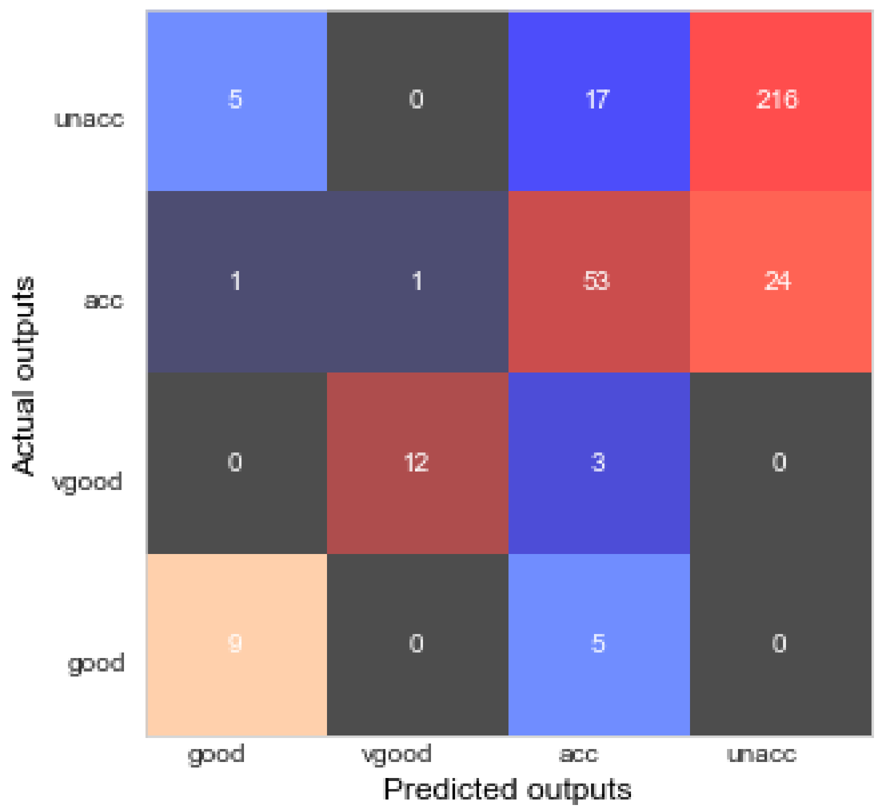
[20]:
```
[<matplotlib.lines.Line2D at 0x170a81f7630>]
```



- The illustrates that training accuracy is increasing with the increasing number of samples

8.2 Confusion Matrix for Actual and Predicted Results

[21]:
```
cm = confusion_matrix(y_test, predictions)

fig, ax= plt.subplots(figsize=(5, 5))
#sns.heatmap(cm, annot=True, fmt="d")
ax.imshow(cm, alpha = 0.7, cmap='flag_r')
ax.grid(False)
ax.set_xlabel('Predicted outputs', fontsize=12, color='black')
ax.set_ylabel('Actual outputs', fontsize=12, color='black')
ax.xaxis.set(ticks=range(10))
ax.yaxis.set(ticks=range(10))
ax.set_ylim(3.5, -0.5)
ax.set_xlim(3.5, -0.5)
for i in range(4):
    for j in range(4):
        ax.text(j, i, cm[i, j], ha='center', va='center', color='white')
plt.yticks([0.1,1.1,2.1,3.1], [ 'unacc', 'acc', 'vgood', 'good'],va='center')
plt.xticks([0.1,1.1,2.1,3.1], [ 'unacc', 'acc', 'vgood', 'good'],va='center')
plt.show()
```

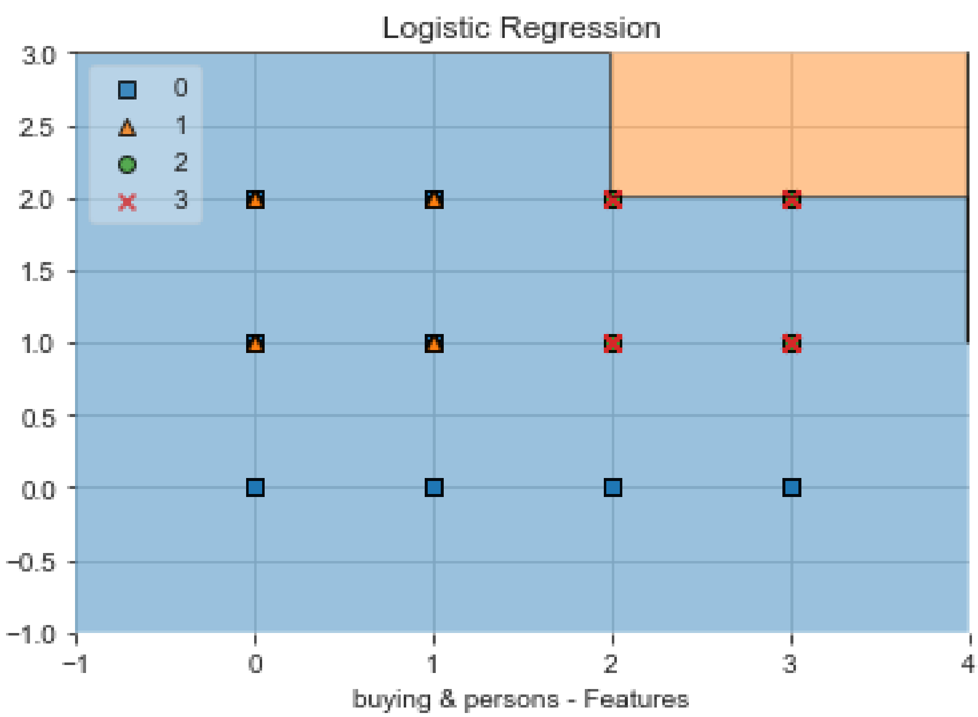


- This is a heatmap that illustrates the confusion matrix with numbers and colors. You can see that the shades of blue and black represent small numbers (like 0, 1, 3 or 5), while red shows much larger numbers (53 and above).

- The numbers on the right diagonal (9, 12, 53, 216) show the number of correct predictions from the test set.

8.3 Plot Decision Boundary

[22]:
```
from mlxtend.plotting import plot_decision_regions
X_train_sample=X_train[['buying', 'persons']].values
y_train_sample=y_train
lgmodel=LogisticRegression(solver='newton-cg', max_iter=1000,␣
 ↪multi_class='multinomial', C=10, random_state=6)
lgmodel.fit(X_train_sample,np.array(y_train_sample))
plot_decision_regions(X_train_sample,np.array(y_train_sample), clf=lgmodel,␣
 ↪legend=2)
plt.xlabel('buying & persons - Features')
plt.title('Logistic Regression')
plt.show()
```



1 9. Predict and display the class label of a car with following attributes : buying, maint, doors, persons, lug_boot, safety as [vhigh,low,4,more,small,med]

Install Lime package to visualize the results

- pip install lime

[23]:
```
import lime
import lime.lime_tabular
```

[24]:
```
feature_names = ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety']
```

[25]:
```
print(car_data['class'].unique())
print(class_names)
```

```
[0 1 2 3]
Index(['unacc', 'acc', 'vgood', 'good'], dtype='object')
```

[26]:
```
explainer = lime.lime_tabular.LimeTabularExplainer(X_train.values,␣
↪feature_names=feature_names,
                                                   class_names=class_names,␣
↪discretize_continuous=True)
```

[27]:
```
#The Explainer Instance #105 #95
for i in [95,105]:
    exp = explainer.explain_instance(X_test.iloc[i], logmodel.predict_proba,␣
↪num_features=6, top_labels=1)
    exp.show_in_notebook(show_table=True, show_all=False)
```

```
<IPython.core.display.HTML object>

<IPython.core.display.HTML object>
```

Above results illustrates that, our model predicts there is 69% of probability that the class is 'unacc' for the row 95 data from test set. and for the row 105 it predicted the class is "vgood"

Let's verify the above results

[28]:
```
# Take a sample
sample = np.array(X_test.iloc[105]).reshape(1,-1)
```

[29]:
```
# Give sample as an input to the logistic regression model
logmodel.predict(sample)
```

[29]:
```
array([2], dtype=int64)
```

The model predicted that the sample belogs to class array 2. let's see what is the value of array number 2.

[30]:
```
car_data['class'].unique()
```

[30]:
```
array([0, 1, 2, 3], dtype=int64)
```

[31]:
```
class_names
```

[31]:
```
Index(['unacc', 'acc', 'vgood', 'good'], dtype='object')
```

hence, the class is predicted as "vgood". Let's verify the input and out from the Test Data.

[32]:
```
print(X_test.iloc[105],'\n \n Target Class Lable:',class_names[y_test.
↪iloc[105]])
```

```
buying      3
maint       3
doors       1
persons     2
lug_boot    2
safety      2
Name: 1673, dtype: int64

 Target Class Lable: vgood
```

1.1 Conclusion

- Data set is a class imbalanced, 70% of the data has only one class(i.e "unacc"), so, we assigned 80% data to the training data set and 20% to the test set for best results
- We achieved 84% accuracy after appliying following paramters on Logistic regression model: {solver='newton-cg', max_iter=1000, multi_class='multinomial', C=10}
- while training we observed the learning rate is increased with the increasing number of samples