The used car market has significantly grown in recent times, with clients ranging from used car dealers and buyers. You are provided with a car evaluation dataset that has features like price, doors, safety, and so on.

Objective: You are required to create a robust model that allows stakeholders to predict the condition of a used vehicle.

Actions to Perform:

Predict the condition of a vehicle based on its features. Plot the most important features. Train multiple classifiers and compare the accuracy. Evaluate the XGBoost model with K-fold cross-validation.

It is a multi-class classification problem. Task is to classify our dataset into 4 classes : Unacceptable, Acceptable, Good, Very-Good.

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

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
data=pd.read_csv('car_evaluation.csv')
```

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

Out[3]:

	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

Independent variables

buying : buying price

maint : maintenance price

doors : number of doors

persons : capacity in terms of persons to carry

lug_boot : the size of luggage boot

safety: estimated safety of the car

Target Variable

Class

In [5]:
```
data.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   class     1728 non-null   object
dtypes: object(7)
memory usage: 94.6+ KB
```

Columns are categorical. We are looking for unique values on each column

In [6]:
```
for i in data.columns:
    print(data[i].unique(),"\t",data[i].nunique())
```

```
['vhigh' 'high' 'med' 'low'] 	 4
['vhigh' 'high' 'med' 'low'] 	 4
['2' '3' '4' '5more'] 	 4
['2' '4' 'more'] 	 3
['small' 'med' 'big'] 	 3
['low' 'med' 'high'] 	 3
['unacc' 'acc' 'vgood' 'good'] 	 4
```

Checking the distribuition of these unique categories among the columns

In [7]:
```
for i in data.columns:
    print(data[i].value_counts())
    print()
```

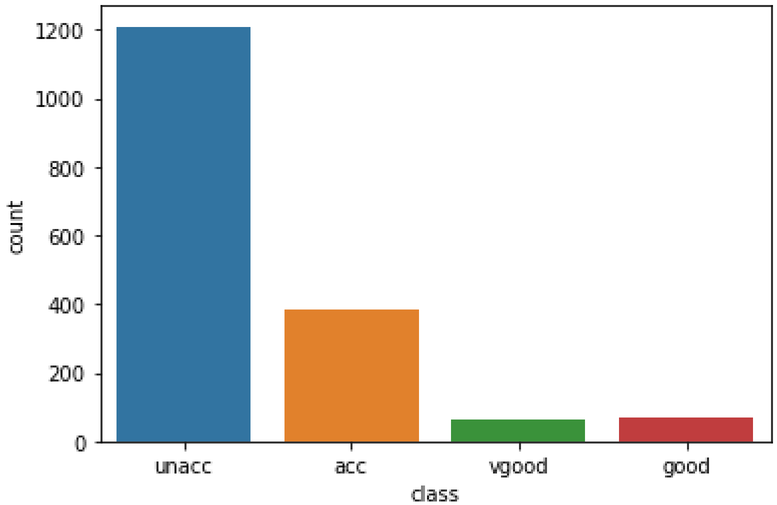
```
med      432
low      432
vhigh    432
high     432
Name: buying, dtype: int64

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

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

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

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

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

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

From the above we can see that all columns except "class" are distribuited equally among data

Class Distribuition

In [8]:
```
sns.countplot(data['class'])
```

Out[8]: <AxesSubplot:xlabel='class', ylabel='count'>



From above we see that 'class' is unbalanced with larger values under 'unacc'. So, this is an unbalanced multiclass classification problem.

Dummy Encoding

In [9]:
```
from sklearn.preprocessing import LabelEncoder
```

In [10]:
```
le=LabelEncoder()
```

In [11]:
```
for i in data.columns:
    data[i]=le.fit_transform(data[i])
```

In [12]:
```
data.head()
```

Out[12]:

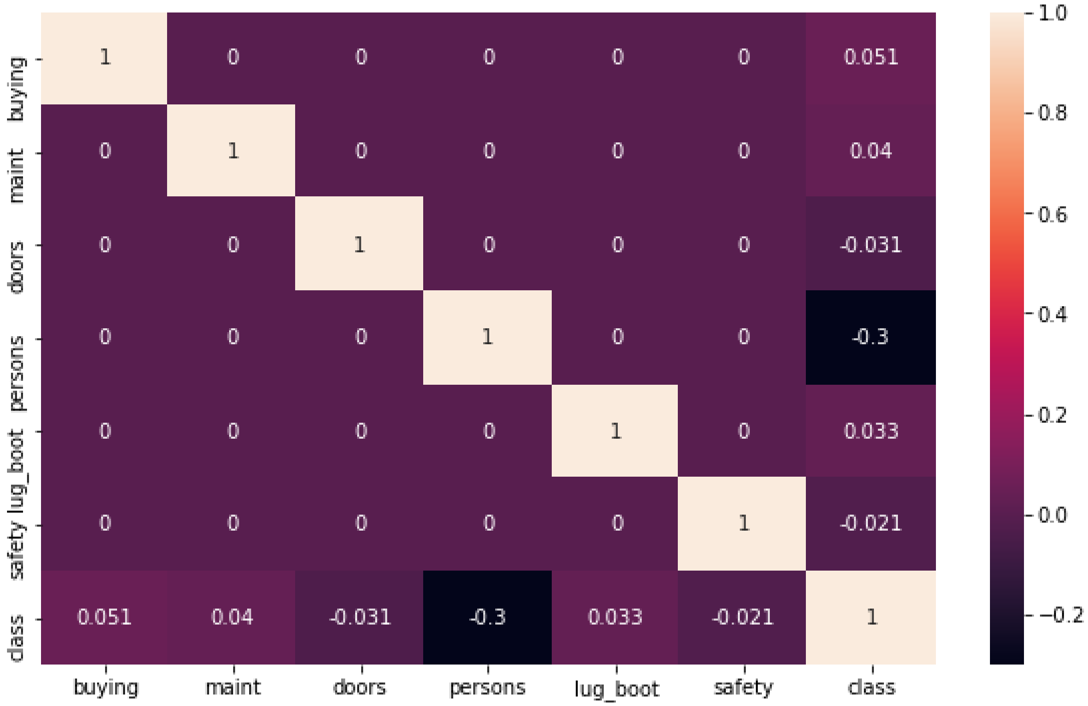
	buying	maint	doors	persons	lug_boot	safety	class
0	3	3	0	0	2	1	2
1	3	3	0	0	2	2	2
2	3	3	0	0	2	0	2

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

Correlation Matrix

In [13]:
```
fig=plt.figure(figsize=(10,6))
sns.heatmap(data.corr(),annot=True)
```

Out[13]: <AxesSubplot:>



Most of columns have a very weak correaltion with "class" so, doing any analysis on them may not give a productive output

In [14]:
```
X=data[data.columns[:-1]]
Y=data['class']
```

In [15]:
```
X.head()
```

Out[15]:

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

In [16]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size=0.3, random_state=42)
```

Model Selection

In [18]:

```
from sklearn.model_selection import learning_curve
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix
```

1. Logistic Regression

In [19]:

```
logreg=LogisticRegression(solver='newton-cg',multi_class='multinomial')
```

In [20]:

```
logreg.fit(X_train,Y_train)
```

Out[20]:

```
LogisticRegression(multi_class='multinomial', solver='newton-cg')
```

In [21]:

```
pred=logreg.predict(X_test)
```

In [22]:

```
logreg.score(X_test,Y_test)
```

Out[22]:

```
0.6647398843930635
```

Logistic Regression is giving a very low accuracy. Will check with other algorithms

2. KNN Classifier

In [23]:
```
knn=KNeighborsClassifier(n_jobs=-1)
```

In [24]:
```
knn.fit(X_train,Y_train)
pred=knn.predict(X_test)
knn.score(X_test,Y_test)
```

Out[24]:
```
0.9017341040462428
```

In [26]:
```
print(classification_report(Y_test,pred))
```

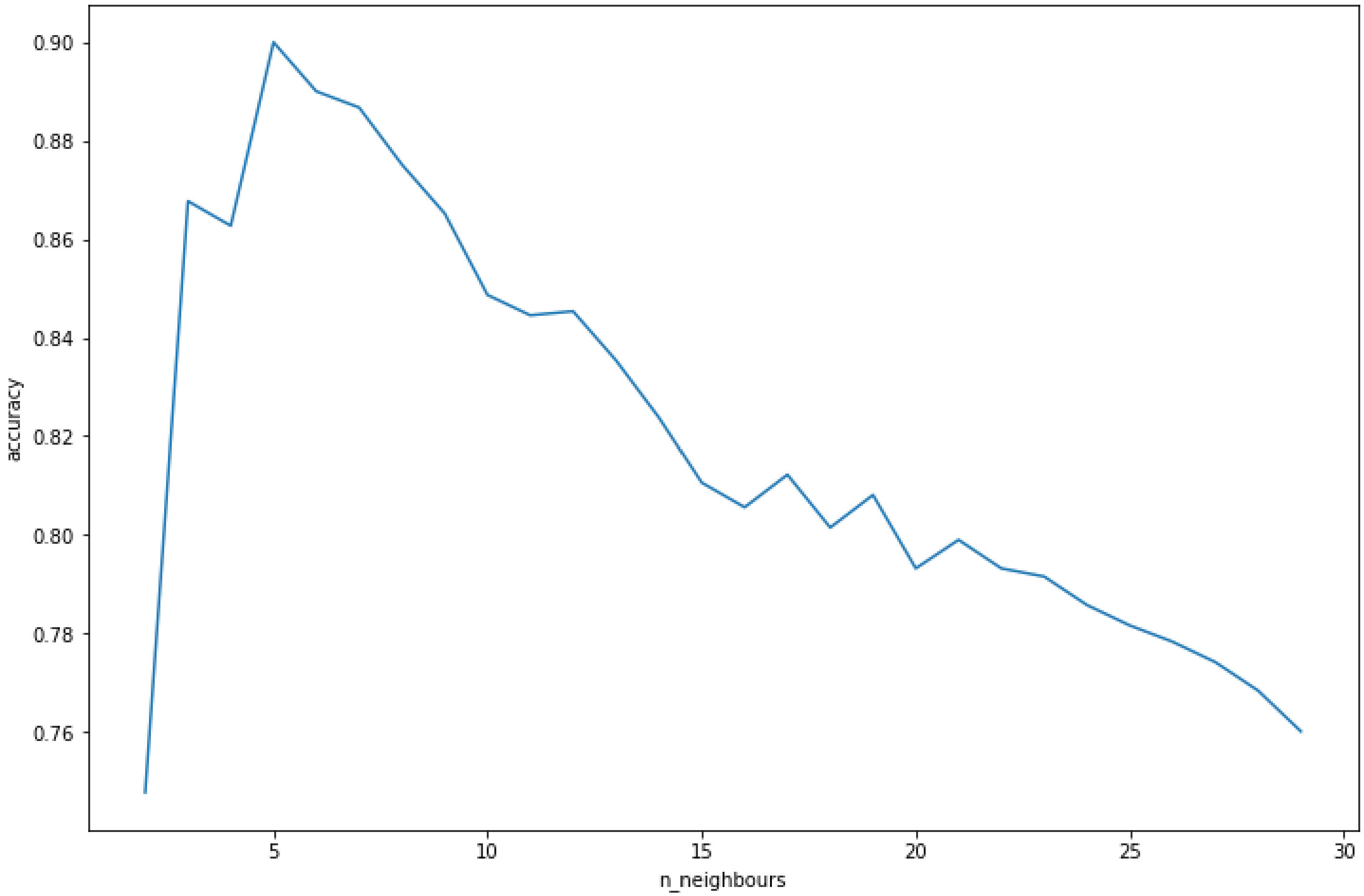
```
              precision    recall  f1-score   support

           0       0.82      0.79      0.80       118
           1       0.77      0.53      0.62        19
           2       0.93      0.99      0.96       358
           3       1.00      0.50      0.67        24

    accuracy                           0.90       519
   macro avg       0.88      0.70      0.76       519
weighted avg       0.90      0.90      0.90       519
```

Accuracy isn't a good criterion to evaluate unbalanced classification, so check 'f1-score' f1-score is 0.9 which is better than previous model

In [27]:
```
avg_score=[]
for k in range(2,30):
    knn=KNeighborsClassifier(n_jobs=-1,n_neighbors=k)
    score=cross_val_score(knn,X_train,Y_train,cv=5,n_jobs=-1,scoring='accuracy')
    avg_score.append(score.mean())
```

In [29]:
```
plt.figure(figsize=(12,8))
plt.plot(range(2,30),avg_score)
plt.xlabel("n_neighbours")
plt.ylabel("accuracy")
#plt.xticks(range(2,30,2))
```

Out[29]:
```
Text(0, 0.5, 'accuracy')
```



n_neighbours=5 is giving better accuracy as well as f1-score for our data So, with KNN Classification we achieve accuracy of 90%

3. Random Forests Classifier

In [30]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score
```

In [33]:
```
rfc=RandomForestClassifier(n_jobs=-1,random_state=51)
```

In [36]:
```
rfc.fit(X_train,Y_train)
print(rfc.score(X_test,Y_test))
print(f1_score(Y_test,rfc.predict(X_test),average='macro'))
```

```
0.9730250481695568
```

```
0.9245337130459484
```

RFC gives 95% accuracy

In [ ]:

```

```