# Machine Learning Classification Algorithms to Car Evaluation Dataset

Team Members-
Eshanth Patyal(21ucs078)
Harsh Gupta(21ucs084)
Harsh Agarwal(21ucs083)
Yash R Khandelwal(21ucs254)

# Contents

# 1.Objective

The project aims to perform preprocessing on a dataset and realise the Different machine learning algorithm learned in the classroom to the data. The dataset contains 6 dependent variables and 1 target variable.

# 2.Specifications of the Dataset

The dataset is taken from the UCI machine learning repository.
Dataset Characteristics-Multivariate
Associated Tasks-Classification
Feature Type -Categorical
Number of Instances 1728
Number of Features 6
**Class Labels-**
Unacc(unaccepetable), acc(accapetable), good, vgood

| **Variable Name** | **Role** | **Type** | **Description** | **Missing Values** | **Variable Name** |
|---|---|---|---|---|---|
| buying | Feature | Categorical | buying price | no | buying |
| maint | Feature | Categorical | price of the maintenance | no | maint |
| doors | Feature | Categorical | number of doors | no | doors |
| persons | Feature | Categorical | capacity in terms of persons to carry | no | persons |
| lug_boot | Feature | Categorical | the size of luggage boot | no | lug_boot |
| safety | Feature | Categorical | estimated safety of the car | no | safety |

# 3.Dataset Description

The Dataset is split into to dataframes-
Df_x=the 6 features
Df_y=the target variable

```
[5]
print(car_evaluation.variables)
```

```
       name     role         type demographic  \
0    buying  Feature  Categorical        None
1     maint  Feature  Categorical        None
2     doors  Feature  Categorical        None
3   persons  Feature  Categorical        None
4  lug_boot  Feature  Categorical        None
5    safety  Feature  Categorical        None
6     class   Target  Categorical        None

                                         description units missing_values
0                                       buying price  None             no
1                           price of the maintenance  None             no
2                                    number of doors  None             no
3                 capacity in terms of persons to carry  None             no
4                           the size of luggage boot  None             no
5                        estimated safety of the car  None             no
6  evaulation level (unacceptable, acceptable, go...  None             no
```

# 4.Libraries Used

```
[1]
pip install ucimlrepo
```

```
Collecting ucimlrepo
  Downloading ucimlrepo-0.0.3-py3-none-any.whl (7.0 kB)
Installing collected packages: ucimlrepo
Successfully installed ucimlrepo-0.0.3
```

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

UCI ML repo imported for the dataset.

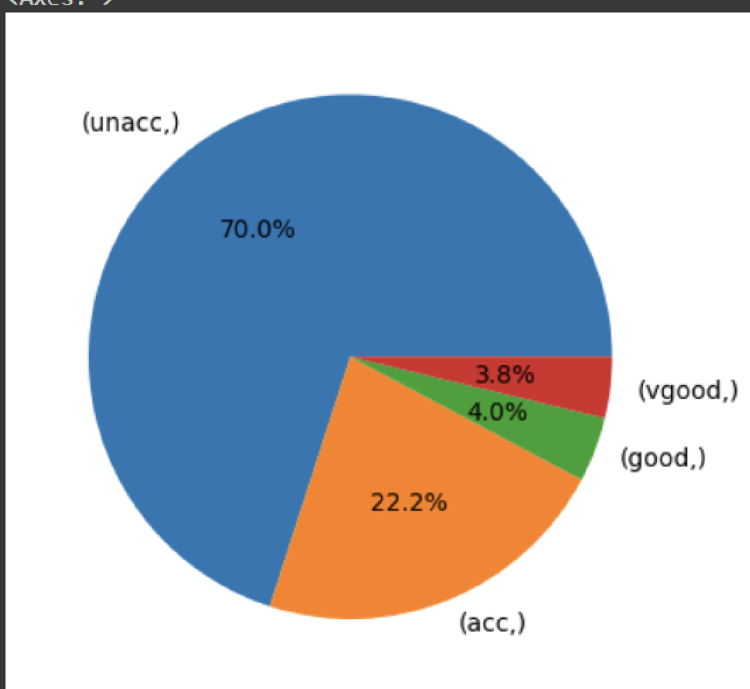
```
from ucimlrepo import fetch_ucirepo

# fetch dataset
car_evaluation = fetch_ucirepo(id=19)

# data (as pandas dataframes)
df_x = car_evaluation.data.features
df_y = car_evaluation.data.targets

# metadata
print(car_evaluation.metadata)

```

```
{'uci_id': 19, 'name': 'Car Evaluation', 'repositor
```

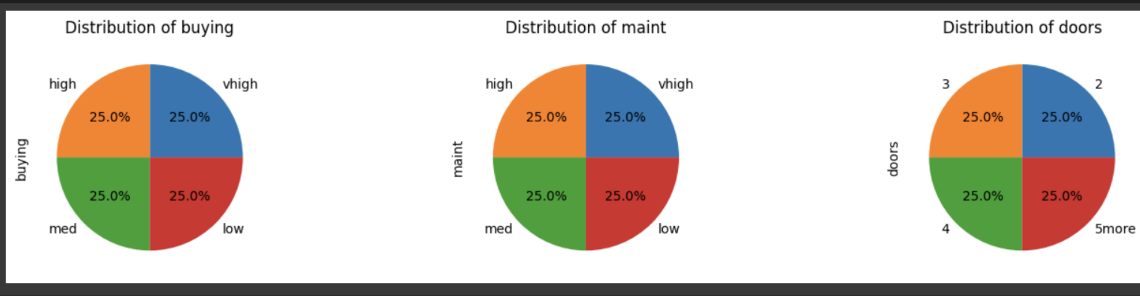
# 5. Statistical Summary

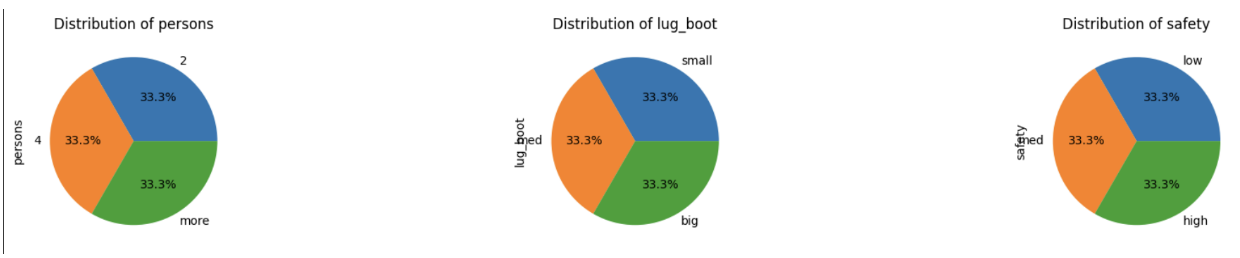
```
df_y.value_counts().plot.pie(autopct='%1.1f%%')
```

```
<Axes: >
```



```
data = ['buying', 'maint', 'doors']

# Set up subplots in a single row
fig, axes = plt.subplots(1, len(data), figsize=(15, 3))

for i, column in enumerate(data):
    df_x[column].value_counts().plot.pie(autopct='%1.1f%%', ax=axes[i])
    axes[i].set_title(f'Distribution of {column}')

# Adjust layout to prevent overlapping
plt.tight_layout()

# Show the plot
plt.show()
```




- Representation of data in class labels is more in unacc class.
- Distribution of all independent variables is evenly spread.

```
1 df_x.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1728 entries, 0 to 1727
Data columns (total 6 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
dtypes: object(6)
memory usage: 81.1+ KB
```

```
[8] 1 df_y.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1728 entries, 0 to 1727
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   class   1728 non-null   object
dtypes: object(1)
memory usage: 13.6+ KB
```

```
[10] 1 df_x.describe().T
```

| | count | unique | top | freq |
|---|---|---|---|---|
| buying | 1728 | 4 | vhigh | 432 |
| maint | 1728 | 4 | vhigh | 432 |
| doors | 1728 | 4 | 2 | 432 |
| persons | 1728 | 3 | 2 | 576 |
| lug_boot | 1728 | 3 | small | 576 |
| safety | 1728 | 3 | low | 576 |

```
[11] 1 df_y.describe().T
```

| | count | unique | top | freq |
|---|---|---|---|---|
| class | 1728 | 4 | unacc | 1210 |

# 6. Data Preprocessing

Data preprocessing is a technique for data mining that involves the translation of raw data into a comprehensible format. In this process, the information used was pre-processed. Since the dataset didn't have any empty values, there was no need to change NaN values.
The issue was Non-Numerical values in many attributes which needed to be changed.

## 6.1 Labelling Data(one hot coding)

We will see that each record has several non-numeric attributes, such as maint, buying, etc., from the available data collection. Generally, learning algorithms need numerical data.
We use the one-hot encoding method, one common way to convert categorical variables to numerical variables.

```python
from sklearn.preprocessing import OneHotEncoder
import pandas as pd

# Concatenate df_x and df_y to perform one-hot encoding on the entire dataset
df = pd.concat([df_x, df_y], axis=1)

# Perform one-hot encoding using pd.get_dummies
df_encoded = pd.get_dummies(df, columns=['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety'])
df_class=pd.get_dummies(df_y)

df_x_encoded = df_encoded.drop(['class'], axis=1)
df_y_encoded = df_class
```

After encoding data looks like-

```python
df_x_encoded[0:5]
```

| | buying_high | buying_low | buying_med | buying_vhigh | maint_high | maint_low | maint_med | maint_vhigh | doors_2 | doors_3 | ... | doors_5more | persons_2 | persons_4 | persons_more | lug_boot_big |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | ... | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | ... | 0 | 1 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | ... | 0 | 1 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | ... | 0 | 1 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | ... | 0 | 1 | 0 | 0 | 0 |

5 rows × 21 columns

```python
[14] df_y_encoded[0:5]
```

| | class_acc | class_good | class_unacc | class_vgood |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 2 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 | 0 |
| 4 | 0 | 0 | 1 | 0 |

# 7.implementing the algorithm

## 7.1 SVM

SVM is effective when dealing with many different decision features feature, making it suitable for datasets with multiple independent variables.
SVM can handle both linear and non-linear decision planes. In this case, where the relationship between features and the target variable is not linear (6 independent variables which are all different), SVM can capture complex patterns.
SVM is robust and is less prone to overfitting in such non-linear relations.

```python
from sklearn.svm import SVC


svc_classifier = SVC()

svc_classifier.fit(X_train, y_train)
y_pred_svc = svc_classifier.predict(X_test)

# Evaluate the model
accuracy_svc = accuracy_score(y_test, y_pred_svc)
classification_report_svc = classification_report(y_test, y_pred_svc)

print(f"SVC Accuracy: {accuracy_svc:.2f}")
print("SVC Classification Report:\n", classification_report_svc)

```

```
SVC Accuracy: 0.97
SVC Classification Report:
               precision    recall  f1-score   support

         acc       0.99      0.89      0.94        83
        good       0.59      0.91      0.71        11
       unacc       1.00      1.00      1.00       235
       vgood       0.84      0.94      0.89        17

    accuracy                           0.97       346
   macro avg       0.85      0.94      0.88       346
weighted avg       0.98      0.97      0.97       346
```

SVM has provided with expectional accuracy in unacc class which has dominated the dataset(70% of the class labels). The same is true for acc and vgood class.

Only good class label (4% of class labels )has low precision.

## 7.2 KNN

KNN is a simple and intuitive algorithm that can be effective when the decision boundaries are not well-defined. It works well when there are regions in the feature space where certain classes are concentrated.

```
[20]
y = df_y
X_train, X_test, y_train, y_test = train_test_split(df_x_encoded, y, test_size=0.2, random_state=42)
```

```
[21]
from sklearn.neighbors import KNeighborsClassifier

# Instantiate the k-Nearest Neighbors Classifier
knn_classifier = KNeighborsClassifier()

# Train the classifier
knn_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred_knn = knn_classifier.predict(X_test)

# Evaluate the model
accuracy_knn = accuracy_score(y_test, y_pred_knn)
classification_report_knn = classification_report(y_test, y_pred_knn)

print(f"KNN Accuracy: {accuracy_knn:.2f}")
print("KNN Classification Report:\n", classification_report_knn)
```

```
KNN Accuracy: 0.88
KNN Classification Report:
               precision    recall  f1-score   support

         acc       0.78      0.75      0.77        83
        good       0.30      0.27      0.29        11
       unacc       0.92      0.99      0.96       235
       vgood       1.00      0.29      0.45        17

    accuracy                           0.88       346
   macro avg       0.75      0.58      0.62       346
weighted avg       0.87      0.88      0.86       346
```

Almost a similar result is seen as good class label is still showing low overall correctness, whereas all other classes seem to do much better in comparison.

## 7.3 Naïve Bayes

It performs well with categorical data and assumes independence between features, making it a good choice when dealing with categorical variables like 'buying', 'maint', etc.

**Naive Bayes**

[23]

```python
from sklearn.naive_bayes import MultinomialNB

nb_classifier = MultinomialNB()
nb_classifier.fit(X_train, y_train)
y_pred_nb = nb_classifier.predict(X_test)

# Evaluate the model
accuracy_nb = accuracy_score(y_test, y_pred_nb)
classification_report_nb = classification_report(y_test, y_pred_nb)

print(f"Naive Bayes Accuracy: {accuracy_nb:.2f}")
print("Naive Bayes Classification Report:\n", classification_report_nb)
```

```
Naive Bayes Accuracy: 0.82
Naive Bayes Classification Report:
               precision    recall  f1-score   support

         acc       0.63      0.54      0.58        83
        good       0.57      0.36      0.44        11
       unacc       0.87      0.97      0.91       235
       vgood       1.00      0.35      0.52        17

    accuracy                           0.82       346
   macro avg       0.77      0.56      0.62       346
weighted avg       0.81      0.82      0.80       346
```

Results from naïve bayes seem to be more like KNN than SVM but the performance of the model is downgraded from KNN.

## 7.4 Logistic Regression

Logistic Regression is easy to interpret and provides probabilities for class assignments. Logistic Regression assumes a linear relationship between the independent variables and the log-odds of the target variable, making it suitable for datasets where such assumptions hold.

**Logistic Regression**

```python
from sklearn.linear_model import LogisticRegression

logreg_model = LogisticRegression()
logreg_model.fit(X_train, y_train)
y_pred_logreg = logreg_model.predict(X_test)

# Evaluate the model
accuracy_logreg = accuracy_score(y_test, y_pred_logreg)
classification_report_logreg = classification_report(y_test, y_pred_logreg)

print(f"Logistic Regression Accuracy: {accuracy_logreg:.2f}")
print("Logistic Regression Classification Report:\n", classification_report_logreg)

```

```
Logistic Regression Accuracy: 0.92
Logistic Regression Classification Report:
              precision    recall  f1-score   support

         acc       0.84      0.82      0.83        83
        good       0.50      0.55      0.52        11
       unacc       0.96      0.97      0.97       235
       vgood       0.94      0.88      0.91        17

    accuracy                           0.92       346
   macro avg       0.81      0.80      0.81       346
weighted avg       0.92      0.92      0.92       346
```

Logistic regression model performs better than KNN and Naïve bayes but not as good as SVM.
The same issues are still consistent (good class label showing low performance) as with other models.

## 7.5 Decision Tree

Decision trees are interpretable and easy to understand, providing insights into feature importance. They can handle both numerical and categorical data without the need for lots of pre-processing. Decision trees can capture non-linear relationships and interactions between features, making them suitable for this dataset.

**Decision Tree**

```
[25]
from sklearn.tree import DecisionTreeClassifier

# Init of Decision Tree
dt_classifier = DecisionTreeClassifier()

# Train the classifier
dt_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred_dt = dt_classifier.predict(X_test)

# Evaluate the model
accuracy_dt = accuracy_score(y_test, y_pred_dt)
classification_report_dt = classification_report(y_test, y_pred_dt)

print(f"Decision Tree Accuracy: {accuracy_dt:.2f}")
print("Decision Tree Classification Report:\n", classification_report_dt)
```

```
Decision Tree Accuracy: 0.97
Decision Tree Classification Report:
              precision    recall  f1-score   support

         acc       0.99      0.90      0.94        83
        good       0.62      0.91      0.74        11
       unacc       0.99      1.00      1.00       235
       vgood       0.82      0.82      0.82        17

    accuracy                           0.97       346
   macro avg       0.86      0.91      0.88       346
weighted avg       0.97      0.97      0.97       346
```

Decision tree model has same overall accuracy as SVM and outperformed all others in this metric.
The model also shows better performance in all classes with only marginally lower performance in unacc class label than SVM.

## 7.6 Random Forest

Random Forest is an ensemble method built on decision trees, providing better generalisation and reducing overfitting.

[26]

```
from sklearn.ensemble import RandomForestClassifier

# Init of Random Forest
rf_classifier = RandomForestClassifier()

# Train the classifier
rf_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred_rf = rf_classifier.predict(X_test)

# Evaluate the model
accuracy_rf = accuracy_score(y_test, y_pred_rf)
classification_report_rf = classification_report(y_test, y_pred_rf)

print(f"Random Forest Accuracy: {accuracy_rf:.2f}")
print("Random Forest Classification Report:\n", classification_report_rf)

```

```
<ipython-input-26-ed49d8c5d23a>:7: DataConversionWarning: A column-vector y
  rf_classifier.fit(X_train, y_train)
Random Forest Accuracy: 0.96
Random Forest Classification Report:
               precision    recall  f1-score   support

         acc       0.97      0.89      0.93        83
        good       0.53      0.82      0.64        11
       unacc       1.00      1.00      1.00       235
       vgood       0.88      0.88      0.88        17

    accuracy                           0.96       346
   macro avg       0.85      0.90      0.86       346
weighted avg       0.97      0.96      0.96       346
```

Random forest Classifier performs better than KNN, Naïve bayes, Logistic regression but fails to trump SVM and Decision Tree.

# 8.Conclusion

Based on the metrics used, we have seen that Decision Tree and SVM proved to be better than all others(even Random Forest was outperformed marginally).

Overall Decision Tree Classifier is better than SVM for the following-

- Overall better performance over all class labels in most metrics.
- Performed better in least represented class label.

The poor performance in good class label could be due to its under-representation in the dataset which caused all models to performance distinctly poor in that class's prediction.

Decision trees perform automatic feature selection by selecting the most important features for splitting and deciding which features in the cars attribute proved more required for the class labels improving the accuracy.