Importing the Data

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt

# importing the data set using pandas library
df = pd.read_csv("car.data")
df.head()
```

Out[1]:

	vhigh	vhigh.1	2	2.1	small	low	unacc
0	vhigh	vhigh	2	2	small	med	unacc
1	vhigh	vhigh	2	2	small	high	unacc
2	vhigh	vhigh	2	2	med	low	unacc
3	vhigh	vhigh	2	2	med	med	unacc
4	vhigh	vhigh	2	2	med	high	unacc

In [2]:
```
# Adding column labels to the dataframe
df.columns = ["buying_price", "maintenance", "doors", "persons", "lug_boot", "safety", "class_value"]
df.head()
```

Out[2]:

	buying_price	maintenance	doors	persons	lug_boot	safety	class_value
0	vhigh	vhigh	2	2	small	med	unacc
1	vhigh	vhigh	2	2	small	high	unacc
2	vhigh	vhigh	2	2	med	low	unacc
3	vhigh	vhigh	2	2	med	med	unacc
4	vhigh	vhigh	2	2	med	high	unacc

In [3]:
```
# obtaining the number of rows and columns from the dataframe
df.shape
```

Out[3]: (1727, 7)

Exploratory Data Analysis and Preprocessing

In [4]:
```
# Checking for any null values in the dataframe
df.isnull().sum()
```

Out[4]:
```
buying_price    0
maintenance     0
doors           0
persons         0
lug_boot        0
safety          0
class_value     0
dtype: int64
```

In [5]:
```
# Checking the distribution of every data column
for col in list(df):
    print(df[col].value_counts())
```

```
high     432
med      432
low      432
vhigh    431
Name: buying_price, dtype: int64
high     432
med      432
low      432
vhigh    431
Name: maintenance, dtype: int64
5more    432
3        432
4        432
2        431
Name: doors, dtype: int64
more    576
4       576
2       575
Name: persons, dtype: int64
med      576
big      576
small    575
Name: lug_boot, dtype: int64
high    576
med     576
low     575
Name: safety, dtype: int64
unacc    1209
acc       384
good       69
vgood      65
Name: class_value, dtype: int64
```

In [6]:
```
# saving the columns needed for the training of the machine learning model
X = df.drop(['buying_price', 'persons'], axis=1)
y = df[['buying_price']]
```

In [7]:
```
# checking the distribution of the label class to ensure that it is not skewed
y.value_counts()
```

Out[7]:
```
buying_price
med             432
low             432
high            432
vhigh           431
dtype: int64
```

In [8]:
```
# checking the sample data
X.head()
```

Out[8]:

	maintenance	doors	lug_boot	safety	class_value
0	vhigh	2	small	med	unacc
1	vhigh	2	small	high	unacc
2	vhigh	2	med	low	unacc
3	vhigh	2	med	med	unacc
4	vhigh	2	med	high	unacc

In [9]:
```
# obtaining the value count of class_value column
X['class_value'].value_counts()
```

Out[9]:
```
unacc    1209
acc       384
good       69
vgood      65
Name: class_value, dtype: int64
```

In [10]:
```
# Preprocessing functions to encode the categorical data
# Manual mapping was used to ensure that the scale/range is reflective of the categorial value

def maintenance_safety_encode(data):
    if data == "low":
        return 1
    elif data == "med":
        return 2
    elif data == "high":
        return 3
    elif data == "vhigh":
        return 4

def lug_encode(data):
    if data == "small":
        return 1
    elif data == "med":
        return 2
    elif data == "big":
        return 3

def lug_encode(data):
    if data == "small":
        return 1
    elif data == "med":
        return 2
    elif data == "big":
        return 3

def door_encode(data):
    if data == "2":
        return 1
    elif data == "3":
        return 2
    elif data == "4":
        return 3
    elif data == "5more":
        return 4

def class_encode(data):
    if data == "unacc":
        return 1
    elif data == "acc":
        return 2
    elif data == "good":
        return 3
    elif data == "vgood":
        return 4
```

In [11]:
```
X.columns
```

Out[11]: Index(['maintenance', 'doors', 'lug_boot', 'safety', 'class_value'], dtype='object')

In [12]:
```
# applying the encoding to the various columns to the respective columns
X['maintenance'] = X['maintenance'].apply(maintenance_safety_encode)
X['doors'] = X['doors'].apply(door_encode)
X['lug_boot'] = X['lug_boot'].apply(lug_encode)
X['safety'] = X['safety'].apply(maintenance_safety_encode)
X['class_value'] = X['class_value'].apply(class_encode)
X.head()
```

Out[12]:

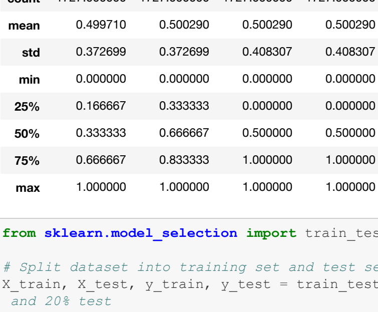
	maintenance	doors	lug_boot	safety	class_value
0	4	1	1	2	1
1	4	1	1	3	1
2	4	1	2	1	1
3	4	1	2	2	1
4	4	1	2	3	1

In [13]:
```
# viewing the distribution of the data to check for skewness
X.hist()
plt.show()
```

In [14]:
```
# reducing the skewness of class value by using the log function
import numpy as np

X['class_value'] = X['class_value'].apply(lambda x: np.log(x+1))
X.hist()

plt.show()
```



In [15]:
```
# applying mean max scaler to ensure all the data are in the same scale
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

X = pd.DataFrame(scaler.fit_transform(X), columns=list(X))

X.describe()
```

Out[15]:

	maintenance	doors	lug_boot	safety	class_value
count	1727.000000	1727.000000	1727.000000	1727.000000	1727.000000
mean	0.499710	0.500290	0.500290	0.500290	0.166253
std	0.372699	0.372699	0.408307	0.408307	0.276485
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.166667	0.333333	0.000000	0.000000	0.000000
50%	0.333333	0.666667	0.500000	0.500000	0.000000
75%	0.666667	0.833333	1.000000	1.000000	0.442507
max	1.000000	1.000000	1.000000	1.000000	1.000000

In [16]:
```
from sklearn.model_selection import train_test_split

# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0) # 80% training
 and 20% test
```

Base Models

In [17]:
```
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, classification_rep
ort, confusion_matrix
```

Logistic Regression

In [18]:
```
from sklearn.linear_model import LogisticRegression

# initialize the model
logreg = LogisticRegression(random_state = 100)

# fitting the training data to the model
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
```

```
/Users/zentan/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataConversio
nWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
  return f(**kwargs)
```

In [19]:
```
# printing the respective accuracy measures of the model
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("F1 Score", f1_score(y_test, y_pred, average="macro"))
print("Precision", precision_score(y_test, y_pred, average="macro"))
print("Recall", recall_score(y_test, y_pred, average="macro"))
```

```
Accuracy: 0.3140655105973025
F1 Score 0.24243704045473213
Precision 0.2565921977627258
Recall 0.3182764284061994
```

KNN

In [20]:
```
from sklearn.neighbors import KNeighborsClassifier

# initialize the model
classifier = KNeighborsClassifier(n_neighbors=5)

# fitting the training data to the model
classifier.fit(X_train, y_train)
y_pred_knn = classifier.predict(X_test)
```

```
/Users/zentan/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:7: DataConversionWarnin
g: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samp
les, ), for example using ravel().
  import sys
```

In [21]:
```
# printing the respective accuracy measures of the model
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("F1 Score", f1_score(y_test, y_pred, average="macro"))
print("Precision", precision_score(y_test, y_pred, average="macro"))
print("Recall", recall_score(y_test, y_pred, average="macro"))
```

```
Accuracy: 0.3140655105973025
F1 Score 0.24243704045473213
Precision 0.2565921977627258
Recall 0.3182764284061994
```

Using pycaret to speed up the process

In [22]:
```
# loading the data into a new dataframe
df_pycaret = pd.read_csv("car.data")

# labeling the dataframe
df_pycaret.columns = ["buying_price", "maintenance", "doors", "persons", "lug_boot", "safety", "class_v
alue"]

# dropping the columns that will not be used for the model
df_pycaret.drop("persons", axis=1, inplace=True)
df_pycaret.head()
```

Out[22]:

	buying_price	maintenance	doors	lug_boot	safety	class_value
0	vhigh	vhigh	2	small	med	unacc
1	vhigh	vhigh	2	small	high	unacc
2	vhigh	vhigh	2	med	low	unacc
3	vhigh	vhigh	2	med	med	unacc
4	vhigh	vhigh	2	med	high	unacc

In [23]:
```
from pycaret.classification import *

# initializing and setting up the pycaret model
clf = setup(df_pycaret, target='buying_price', session_id=1)
```

	Description	Value
0	session_id	1
1	Target	buying_price
2	Target Type	Multiclass
3	Label Encoded	high: 0, low: 1, med: 2, vhigh: 3
4	Original Data	(1727, 6)
5	Missing Values	False
6	Numeric Features	0
7	Categorical Features	5
8	Ordinal Features	False
9	High Cardinality Features	False
10	High Cardinality Method	None
11	Transformed Train Set	(1208, 18)
12	Transformed Test Set	(519, 18)
13	Shuffle Train-Test	True
14	Stratify Train-Test	False
15	Fold Generator	StratifiedKFold
16	Fold Number	10
17	CPU Jobs	-1
18	Use GPU	False
19	Log Experiment	False
20	Experiment Name	clf-default-name
21	USI	eeb0
22	Imputation Type	simple
23	Iterative Imputation Iteration	None
24	Numeric Imputer	mean
25	Iterative Imputation Numeric Model	None
26	Categorical Imputer	constant
27	Iterative Imputation Categorical Model	None
28	Unknown Categoricals Handling	least_frequent
29	Normalize	False
30	Normalize Method	None
31	Transformation	False
32	Transformation Method	None
33	PCA	False
34	PCA Method	None
35	PCA Components	None
36	Ignore Low Variance	False
37	Combine Rare Levels	False
38	Rare Level Threshold	None
39	Numeric Binning	False
40	Remove Outliers	False
41	Outliers Threshold	None
42	Remove Multicollinearity	False
43	Multicollinearity Threshold	None
44	Remove Perfect Collinearity	True
45	Clustering	False
46	Clustering Iteration	None
47	Polynomial Features	False
48	Polynomial Degree	None
49	Trignometry Features	False
50	Polynomial Threshold	None
51	Group Features	False
52	Feature Selection	False
53	Feature Selection Method	classic
54	Features Selection Threshold	None
55	Feature Interaction	False
56	Feature Ratio	False
57	Interaction Threshold	None
58	Fix Imbalance	False
59	Fix Imbalance Method	SMOTE

In [24]:
```
# training and obtaining performance of models
compare_models()
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
nb	Naive Bayes	0.3179	0.5404	0.3180	0.3047	0.2500	0.0900	0.1123	0.0050
ada	Ada Boost Classifier	0.3162	0.5757	0.3153	0.3701	0.3063	0.0863	0.0903	0.0230
ridge	Ridge Classifier	0.3129	0.0000	0.3123	0.3435	0.2963	0.0824	0.0875	0.0060
lda	Linear Discriminant Analysis	0.3121	0.5750	0.3111	0.3702	0.2985	0.0808	0.0855	0.0070
lr	Logistic Regression	0.3055	0.5763	0.3050	0.3311	0.2925	0.0726	0.0764	0.6180
svm	SVM - Linear Kernel	0.2848	0.0000	0.2844	0.2906	0.2422	0.0454	0.0572	0.1500
gbc	Gradient Boosting Classifier	0.2583	0.5740	0.2586	0.2818	0.2581	0.0100	0.0097	0.0710
qda	Quadratic Discriminant Analysis	0.2517	0.0000	0.2500	0.0633	0.1012	0.0000	0.0000	0.0100
knn	K Neighbors Classifier	0.1963	0.4402	0.1965	0.1978	0.1934	-0.0719	-0.0726	0.1860
lightgbm	Light Gradient Boosting Machine	0.1871	0.4956	0.1877	0.1906	0.1864	-0.0842	-0.0849	2.2510
xgboost	Extreme Gradient Boosting	0.1722	0.4595	0.1728	0.1768	0.1722	-0.1041	-0.1051	1.1130
dt	Decision Tree Classifier	0.1681	0.4297	0.1689	0.1545	0.1553	-0.1091	-0.1122	0.0050
et	Extra Trees Classifier	0.1656	0.4248	0.1664	0.1537	0.1538	-0.1124	-0.1155	0.0990
rf	Random Forest Classifier	0.1648	0.4354	0.1653	0.1726	0.1666	-0.1140	-0.1147	0.1060

Out[24]: GaussianNB(priors=None, var_smoothing=1e-09)

In [25]:
```
# selecting the best performing model and saving it
model = create_model('nb')
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.3388	0.5487	0.3417	0.3347	0.2804	0.1209	0.1529
1	0.2975	0.5287	0.3000	0.3278	0.2266	0.0659	0.0847
2	0.2479	0.4794	0.2500	0.3039	0.1917	0.0000	0.0000
3	0.2975	0.5281	0.3000	0.2923	0.2195	0.0659	0.0872
4	0.3223	0.5242	0.3241	0.3308	0.2774	0.0959	0.1106
5	0.3223	0.5429	0.3156	0.2422	0.2231	0.0899	0.1207
6	0.3636	0.6000	0.3621	0.3274	0.3068	0.1471	0.1786
7	0.3140	0.5295	0.3087	0.2793	0.2437	0.0791	0.1000
8	0.3167	0.5451	0.3181	0.2925	0.2467	0.0896	0.1093
9	0.3583	0.5777	0.3601	0.3160	0.2838	0.1452	0.1793
Mean	0.3179	0.5404	0.3180	0.3047	0.2500	0.0900	0.1123
SD	0.0315	0.0307	0.0311	0.0276	0.0342	0.0408	0.0498

In [26]:
```
# tuning the model to improve performance of classification results
tuned_nb = tune_model(model)
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.3388	0.5487	0.3417	0.3347	0.2804	0.1209	0.1529
1	0.2975	0.5287	0.3000	0.3278	0.2266	0.0659	0.0847
2	0.2479	0.4794	0.2500	0.3039	0.1917	0.0000	0.0000
3	0.2975	0.5281	0.3000	0.2923	0.2195	0.0659	0.0872
4	0.3223	0.5242	0.3241	0.3308	0.2774	0.0959	0.1106
5	0.3223	0.5429	0.3156	0.2422	0.2231	0.0899	0.1207
6	0.3636	0.6000	0.3621	0.3274	0.3068	0.1471	0.1786
7	0.3140	0.5295	0.3087	0.2793	0.2437	0.0791	0.1000
8	0.3167	0.5451	0.3181	0.2925	0.2467	0.0896	0.1093
9	0.3583	0.5777	0.3601	0.3160	0.2838	0.1452	0.1793
Mean	0.3179	0.5404	0.3180	0.3047	0.2500	0.0900	0.1123
SD	0.0315	0.0307	0.0311	0.0276	0.0342	0.0408	0.0498

In [27]:
```
# creating test data for prediction
test = pd.DataFrame([['High', '4', 'Big', 'High', 'Good']])
test.columns = ["maintenance", "doors", "lug_boot", "safety", "class_value"]

test.head()
```

Out[27]:

	maintenance	doors	lug_boot	safety	class_value
0	High	4	Big	High	Good

In [28]:
```
# using the model to predict the test data
unseen_predictions = predict_model(tuned_nb, data=test)

unseen_predictions.head()
```

Out[28]:

	maintenance	doors	lug_boot	safety	class_value	Label	Score
0	High	4	Big	High	Good	low	0.9841

The predicted Buying Price would be low